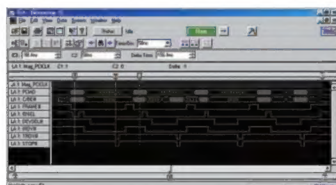
2004
1
Contents



[表紙デザイン：㈱プランニング・ロケッツ]







特集

PCI-X対応デバイスの設計からドライバ/PCI BIOSの作成まで

基礎からわかる PCI & PCI-X活用技法

Cover Story PCI & PCI-X Application Technique from the Basics 39

Interface

2004 1

Contents

連載「XScaleプロセッサ徹底活用研究」,「TOPPERSで学ぶRTOS技術」,「開発技術者のためのアセンブラ入門」,「プログラミングの要」,「やり直しのための信号数学」は,お休みさせていただきます.

アジア最大のエレクトロニクス総合展

CEATEC JAPAN 2003



北村 俊之

「ユビキタス・コミュニティ，次へ始動！」をテーマに「CEATEC JAPAN 2003」が10月7日(火)～11日(土)の5日間，日本コンベンションセンター(幕張メッセ)で開催された．主催は情報通信ネットワーク産業協会(CIAJ)，(社)電子情報技術産業協会(JEITA)，(社)日本パーソナルコンピュータソフトウェア協会(JPSA)である．以前開催されていた「エレクトロニクスショー」と「COM JAPAN」を統合して2000年から始まった本展示会は，今年で第4回目を迎え，667社/団体が2552小間を出展した．

● 電子部品・デバイス＆装置 ステージでは……

このステージでは，半導体や基板など最新の電子技術を中心に展示されており，携帯電話やディジタルカメラ，車載用システムなど，幅広い応用事例が多数展示されていた．

ミツミ電機では，小型カメラモジュール「CMV-81BX」の展示が行われていた．同製品は，一般携帯電話，車載モニタおよび小型センサなどの用途に適するという．小型VGA仕様として開発されており，従来サイズ(10mm)に対して，基本性能を同等に，8.25mmサイズの小型化を実現しているのが特長であるという．また，同ブースでは参考出品ではあったが，パノラミックカメラモジュール「PCM-CP001」(写真1)の展示も行われていた．こちらは，レンズを駆動することなく，360°の環状画像撮影が可能な製品で，店内監視や交差点監視，テレビ会議などの用途に適しているという．



〔写真1〕パノラミックカメラモジュール PCM-CP001

最近のサーマルプリンタの小型化には，目を見張るものがある．シチズンCBMのブースでは，小型のサーマルプリンタ「CT-S300」(写真2)の展示を行っていた．145×195×121mmの外形寸法で，72mm/576ドットの印字幅を実現している．また，最大で100mm/sの高速印字が可能だという．横置き，縦置き，壁掛け(オプション)が可能で，レシート発行，計測器データの印刷やキッチンプリンタなどの用途を目的としているという．



〔写真2〕シチズンの小型サーマルプリンタ CT-S300

東芝では，SoC技術を集約したSバンド(2.6GHz帯)を使用したモバイル放送用チップセットの展示を行っていた(写真3)．こちらは，移動体，個人向けの衛星ディジタル放送サービスを受信するための端末に最適なチップセットだという．また，同社のRISCソリューションを利用した，マルチメディアホームネットワークシステムの展示も行われていた．



〔写真3〕東芝のモバイル放送用チップセットの展示

アークでは，磁界発電による非接触型充電装置を展示していた．置くだけで充電が可能となる「おいとけ電池」および「充電パッド」(写真4)で，同製品を利用すれば，リモコンなどの電気機器から電池を取り外すことなく，充電が可能になるという．



〔写真4〕アークの磁界発電による非接触型充電装置

● デジタルネットワーク ステージでは……

こちらのゾーンは，家電，民生機器が中心の構成となっており，大手メーカーが巨大ブースを構えて，来場者の注目を集めていた．とくに，来るべきディジタル放送を視野に入れたコンセプトモデルが多く出展されていた．全般的には，先頃開催された「WPC Expo 2003」と変わりばえしないというのが，個人的な印象として残った．KDDIブースで来場者の注目を集めていた，斬新なデザインの「INFOBAR」(写真5)は，直線的なフォルムを引き立てる独創的なキーデザインと11mm，87gという小型，軽量ボディが大きな特徴であるという．もちろん，4倍ズームのCCDカメラ，ムービーメール，BREW対応，GPSなどの従来機能も継承されている．



〔写真5〕KDDIのINFOBAR

ソニーブースの注目の的は，本展示会で初めてお目見えした「PSX」(写真6)．「DESR-7000」モデルは，250Gバイトのハードディスクを搭載し，最大325時間の録画が可能であるという．録画だけではなく音楽や静止画像，ゲーム機能を融合させた．さらにネットワーク接続により，さまざまな機能の追加，更新ができるなど，まさにソニーらしさを全面に押し出した製品といえよう．



〔写真6〕ソニーのPSX

三洋電機では，3Gコンセプトモデルとして「地上デジタルテレビスタイリッシュタイプ」の展示を行っていた．こちらは，ワンアクションでテレビとの切り替えが可能，メールやWebに利用可能なサブディスプレイの搭載などを特徴としている．また，同社初となるディジタルムービーカメラ「Xacti〔DMX-C1(S)〕」(写真7)も来場者の関心が高い製品であった．こちらは最近流行の，高画質な動画と静止画像を一台でこなせるモデルである．有効320万画素で4画素混合動画対応の原色CCDを搭載し，MPEG-4方式(VGAサイズ)の動画をサポートしている．光学5.8倍ズームレンズを搭載し，ディジタルズームと併用で最大60倍のズームが可能となっている．149cc，153gという小型，軽量なサイズも魅力の一つであるという．ただし，レンズの下にストロボがついているのは，構造上問題のような気がした．



〔写真7〕三洋のXacti

東芝ブースでは，実際に稼動している「モバイル放送端末」を出展していた(写真8)．また，同ブースで注目を集めていたのが，ノートパソコン用の「燃料電池」である．こちらは，参考出品とのことであったが，現在約5時間稼動と10時間稼動タイプのものを商品化に向けて準備中だという．



〔写真8〕東芝のモバイル放送端末

シャープのブースでは，先頃発表された「オーディオ回路内蔵システム液晶」のデモを行っていた．同技術では，CGシリコン技術により，液晶パネルのガラス基板上にオーディオ回路を一体形成している．モバイル機器の小型化，薄型化のニーズに応えるという．同社では，この技術を用いて，映像表示と音声出力機能を一体化した「LCDパネルスピーカ」とステレオ対応の「スピーカ外付けタイプ」を開発している．

ハッカーの常識的見聞録

広畑 由紀夫



今月の常識

「Longhornを目にしよう！」

☆米国で開催されたProfessional Developers Conference 2003を皮切りに，いよいよ次期クライアント用Windowsである Longhornが公開されます．今回は，日本での公開開始に向けてまとめておきましょう．

● Longhornがやって来る!!

Longhornとは，Windows XP後継OSの開発コードです．ここ数年噂が一人歩きし，多くの憶測と話題を提供してくれたものなので，皆さんも名前についてはよく耳にしていることと思います．2003年春に一部情報が解禁され，どのような製品をめざすかといった一例も公開されましたが，いよいよ10月26日～28日に米国で開かれた「Professional Developers Conference 2003」においてβ版が公開され始めました．

● SQL Server「Yukon」とは？

従来，データベースはそれぞれのプロセッサ向けに最適化されたネイティブコードによって動作していましたが，Yukonでは.NET Framework上のCLR（共通言語ランタイム）が組み込まれる予定です．現在のところ，CLRは目立った機能ではないものの，「Yukon」が発売され，導入が開始される頃のWindowsクライアントには，かなり普及していることと思われます．このことは，データベースサーバとしての機能だけではなく，コードの再利用やコードそのもののセキュリティ強化，CLRによる他言語プログラマとの連携の強化などの，多くの開発レベルにおける協調に，データベースサーバ自体が組み込まれていくことを意味しているのでしょう．

● 次期Visual Studio.NET「WHIDBEY」「ORCAS」

まず「WHIDBEY」については，SQL Server「Yukon」向け開発が統合され，Longhornまでのクライアント開発環境がひととおりそろうようです．データベースアクセスにしても，従来のアプリケーション開発などは個々のコンポーネントに対して，ラッパクラスなどで実装し直したりしていたものですが，IDE上で多くの操作がコードを記述することなく自動生成され，開発者の負担が軽減されることでしょう．近年とくにOfficeやデータベースへの依存が，企業内インフラが整うにつれて高まってきていることもあり，そうした方面へのさらなる開発者への負担の軽減は望ましいことです．

「ORCAS」は2005年以降の予定となっています．Longhorn向け開発ツールとして，さらに強化されたマネージインターフェース，拡張UIなどをサポートする開発ツールとなるようです．まだ「WHIDBEY」が正式になっていないまでも，Longhornが視野に入ってきた現在，その開発ツールの公開時期は非常に興味をかきたてられます．

● 統合されていく開発環境

Longhornの登場までに，現在個別に分かれている開発環境やライブラリなどの統合化，さらにはサーバ自身へのプログラムアクセス方法についても，IDEから簡単に行えるような統合化によって，より少人数のエンジニアによる短期間開発へ導いてくれるものと期待しています．また，ツールによる自動生成の強化や分散編集によって，エンジニアのスキルに応じた分業の高度化も果たせるのではないかと考えています．Office製品においても「Visual Studio Tools for Office 2003」を使用することにより，Word2003，Excel2003へ，マネージドコードによって高度なカスタム化を図れるようになり，テンプレートレベルではできなかった業務クライアントが比較的簡単に開発できるものと考えています．さらには，マネージドコードのため，コード自身の再利用や今後のプラットホーム変更に対する柔軟性も，64ビットプラットホームが視野に入ってきた現在は重要なことではないかと考えます．

● 日本でも公開間近！

日本国内では2003年12月9日と10日に，「ホテル グランパシフィック メリディアン」において「.NET Developers Conference 2003」が開催されます．この「.NET Developers Conference 2003」にて公開されるとのことなので，参加申し込みをされた方はぜひ当日目にしていただきたいと思います．もちろん筆者も参加します．これらのカンファレンス後は，MSDN会員向けの提供などが会員レベルに応じて行われると思われますが，カンファレンスに参加して情報を得て活用していきたいものです．

- Professional Developers Conference 2003公式ページ
  http://www.event-info.jp/netdc/default.htm
- 開発者向けロードマップ（英語）
  http://www.microsoft.com/japan/msdn/vstudio/productinfo/roadmap.asp
- SQL Server“Yukon”
  http://www.microsoft.com/japan/sql/evaluation/yukon.asp

ひろはた・ゆきお OpenLab.

BSD Conference Japan 2003

■ 日時：2003年10月18日（土）
■ 場所：BIZ新宿（東京都新宿区）

BSDカンファレンス2003実行委員会により，「BSDの今を伝える」を目的としたカンファレンスが開催された．

奈良先端科学技術大学院大学の砂原秀樹氏による「Beyond the BSD：次世代のOS開発へ向けて」では，BSDの発展に深く関わってきた氏の立場から，「BSDはcshやvi，Xをはじめとした計算機利用環境を改善したこと，仮想記憶やTCP/IP，各種ファイルシステムなどの新しいOS機能を研究するためのプラットホームとして役立ってきた」ということなどが語られた．そのうえで，BSDコミュニティに望むこととして「BSDの枠にはまらずに新しいものに挑戦してほしい」というエールが送られた．また，砂原氏自身は「そろそろOSをフルスクラッチで作り直すべきではないか」との意見をもち，分散環境の自動把握と適応/入出力の位置透過性/実行に最適なノードの自動選択/自己組織性などの機能をもったOS「SiON」を開発しているとのことだった．

その他の講演として，榮樂英樹氏（筑波大学情報学類），新城靖氏（筑波大学電子・情報工学系）の「ユーザレベル BSD のための軽量 VM」，塩崎拓也氏（NetBSD Project，Citrus Project）の「Citrus iconv の実装」，曽田哲之氏（（株）SRA）の「Scheduler Activations とは？～pthreads for NetBSD～」など．



奈良先端科学技術大学院大学の砂原秀樹氏

また，従来のFreeBSDやNetBSDといった，「*BSD」としてよく知られているOSだけでなく，Mac OS Xも取り上げられていた点が目を引いた．

Linux Kernel Conference 2003

■ 日時：2003年10月9日（木）～10日（金）
■ 場所：青山ダイヤモンドホール（東京都港区）

Linuxカーネルの最新動向を取り上げたカンファレンスが開催された．

安定版カーネルの主任メンテナであるDigio社のAndrew Moron氏による「カーネル2.6の新機能概論」では，スケーラビリティの増加などの性能改善，2.6での新デバイスのサポートなどが中心に取り上げられた．

カーネル2.6において，組み込み分野において期待されるのはプリエンティブルカーネルの正式採用である．システムコールを実行しているときにもコンテキストスイッチが可能になり，応答性が改善される．また，μClinux（MMUをもたないプロセッサでも動作可能にしたLinux）が本家カーネルに統合される予定である．MC68000/ARM7/H8などでも動作させることが可能で，小規模の組み込み機器へのLinux適用の道が拓かれたといえる．

組込みソフトウェアシンポジウム2003 (ESS 2003)

■ 日時：2003年10月16日（木）～17日（金）
■ 場所：機械振興会館（東京都港区）

（社）情報処理学会により，「組込みソフトウェア研究・開発のリエンジニアリングを目指して」と題したシンポジウムが開催された．

トヨタ自動車（株）の林和彦氏による基調講演「自動車産業におけるソフトウェア開発」では，自動車に搭載されるECU（Electronic Control Unit）の個数はクラウンクラスで60以上，06年車種では70～80と急激に増加しているとの現状を挙げ，これらを接続する配線とソフトウェアも増加し，これを削減することが急務であるとのことだった．このような状況で，ECUハードウェアとソフトウェアの分離，アプリケーションとプラットホームを分離することにより，ハードウェアの上に統一的なプラットホームを載せ，その上にアプリケーションを乗せるという方向へ転換することで解決したいとのことであった．

Googleカフェ

■ 日時：2003年10月8日（水）～10日（金）
■ 場所：品川Virgin（東京都品川区）

検索サービスを提供するグーグル（株）が品川の喫茶店「VIRGIN CAFE」で同社のサーチエンジンの普及を目的としたイベント「Googleカフェ」を開催した．

同カフェではクイズが出題され，店内に設置されたノートPCでGoogleへアクセスして回答を検索することを通じてGoogleに慣れ親しんでもらいたい，とのことだった．また，店内には「Googleアドバイザ」が常駐し，検索に際して適切なアドバイスを与えていた．

同社が今回のイベントを企画した趣旨としては，「イメージ検索の知名度が低いこと，効率的な検索方法が広まっていないこと（一つしかキーワードを指定しないなど）を改善したい」とのことであった．



マーケティングディレクタのDoug Edwards氏

移り気な情報工学

ITもの作りの原点

山本 強

前回は「ビットの化石」と題してCP/M 1.4復元計画を実行し，エミュレータ上での動作が確認できた話をした．

その時点ではディスク内容のバイナリイメージを復元し，ネットで入手したエミュレータで個々のユーティリティの動作までは行ったが，その先，つまり完全なるCP/M 1.4の復元まではたどり着いていない．8080のエミュレータを使えばすぐにでもブートできそうなものだが，CP/M 1.4は77トラック×26セクタという8インチ片面単密度のFDの物理フォーマットを前提に作られているので，FDDのエミュレータも作らなければならない．

それ以前に，データリカバリを依頼した結果として帰ってきたのは，マスターディスクのコンテンツをファイルとして解読してWindowsのファイルシステムに変換してくれたものだった．データ復元という意味ではこれで良いのだが，私がやりたかったのはシステムの復元だったので，改めてディスクの物理復元を追加発注している所である．たかがCP/Mなのだが，0から始めるとなるとこれがなかなか疲れるのである．何でもそうだが，何もない所から始めることはたいへんなのである．

遠くなる技術の原点（?!）

20世紀末からの数年間だけで，IT革命，ITバブル崩壊，そしてユビキタスや電子タグをキーワードとし，組み込み系，IT，もの作りの復興と持ち上げられたり叩かれたりと忙しいIT業界なのだが，とりあえずハード，ソフトの開発手法高度化はどんどん進んでいる．開発を効率化しつつ信頼性をあげるためには，開発環境の高度化が必須なわけで，それがあっての現代IT社会が成立しているのは間違いない．

しかし，何かが変だと思うことがある．かつて開発スタイルが原始的だったころの開発に要求された技術常識と，今の抽象化された開発スタイルに要求される技術常識が変わってきているように見えるのである．

かつて手作りのパソコンにCP/Mを入れるときには，FDの物理フォーマットまで知っていることが常識だったのだが，DOS/Vパソコン時代になるとFDの常識はケーブルの接続法までで止まってしまう．ソフトウェアもVB世代になってソフト開発の重要なノウハウは，いかに多くのコンポーネントの使い方を知っているかということになってきている．

それ以前に，かつては組み込み系の話をするときに，ハード，ソフトという分化があいまいで，ハードを設計する人が，ソフトも作っていた．しかし，それでは効率が悪いということで，開発スタイルが階層化し，需要の多い上層担当の技術者が求められているのだから，そういう層に特化した技術者教育がもてはやされるのも当然といえば当然なのである．

原点を忘れると何がおこるか？

有名なところでは，インテル8080の初版でのGNDパターンの設計ミスの話がある．かつてインテルが8080を作ったとき，論理的にはエラーがなかったにも関わらず，GNDのパターンが細かったために，ワーストケースでは論理0の電圧が十分に低下しなくなり，動作条件が厳しくなってしまうということがあった．レイアウト，回路設計，論理設計が分化した結果，全体を見る視点で盲点が出たということか．LSIもオームの法則に縛られるという原点を忘れてはいけない．

ソフトウェア開発はスタート地点が高レベルになると，逆に自由度が低下する．高レベルな開発システムで作るとできあがるプロダクトの外見や機能が似てくるのである．もちろん，開発ツールを提供する側は，デザイン変更やコンポーネントそのものの開発もサポートするのだが，使う側は開発ツールに付属するサンプルコードの改造でできることに限定して使っていることが多い．

以前，ファイル圧縮解凍ツールを探したときのことなのだが，あるサイトでフリーソフトウェアがたくさんならんでいるのを見て，まだまだ日本のソフト開発力も捨てたものではないと思ったことがある．しかし，実際にダウンロードして走らせると肝心の圧縮・解凍のエンジンはどれも同じDLLを使っていて，違うのはユーザーインターフェースという皮の部分だけだった．これらツール開発の原点はアルゴリズムではなく，DLLであるらしい．

計算機業界ではずいぶんと昔にハードとソフトの分化がなされており，今では教育体系もまったく異なっている．たしかに情報処理としてのハードとソフトは最終製品のレベルで別物として取り扱われる社会通念が同時にできたため，自然と受け入れられているともいえる．

問題は，元祖日本組み込み系である．これは最終製品が「物」であり，ソフトとハードが不可分なのである．今一度，IT物づくりの原点を確認しておく必要があると感じている．

やまもと・つよし　北海道大学大学院工学研究科電子情報工学専攻
計算機情報通信工学講座 超集積計算システム工学分野

PCI-X対応デバイスの設計から
ドライバ/PCI BIOSの作成まで

基礎からわかる PCI & PCI-X活用技法

次世代の汎用拡張バスとして，PCI-Expressが期待されている．しかし，PCI-Expressを搭載したシステムが普及するには，まだ多少の時間がかかりそうである．そこで，サーバ用途のPCでは，いますぐにでも使える高速バスとして，PCI-Xが標準で搭載されるようになっている．ギガビット級のネットワークやRAIDのストレージとの接続を考えると，通常のPCIバスでは帯域が足りなくなるためだ．

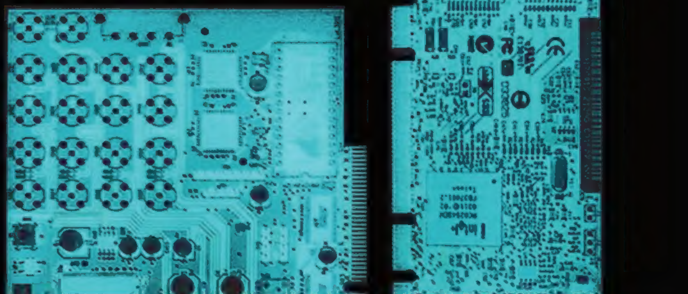
PCI-Xは，従来のPCIと物理的なコネクタやピン配置などに互換性がある．さらに，PCI-XシステムにPCIボードを差し込んで使うこともできるし，その逆も可能であるなど，互換性を重視して規格化された高速バスである．

そこで，今回の特集では，いますぐに使える高速バスであるPCI-Xを取り上げる．まずPCIおよびPCI-Xのプロトコルについての基礎知識を解説した後，FPGAによりPCI-X対応デバイスを設計する．また，バスマスタPCIデバイスに対応したWindowsドライバ作成方法についても解説する．

さらに，組み込み機器向けのPCI BIOSの作成方法や，PCI拡張ROMのプログラムの作り方についても解説する．

そして，今回設計したPCI-X対応デバイスのHDLソースコードはもちろん，Windowsデバイスドライバ，PCI BIOS，PCI拡張ROM，新バージョンのPCIデバッグライブラリ for DOSなど，特集に関連したソースファイルを本誌の付属CD-ROM InterGiga No.32に収録する．

Contents



PCIバスを取り巻く現状

井倉 将実

● PCIはスタンダード

はじめてPCIという名前で仕様書が発行されたのは，1992年のことだそうです．すでに10年以上の歴史があるわけで，数え切れないほどの多くの企業が，チップセット/マザーボード/デバイスなど，PCIバスを採用したシステムを開発/製造/出荷しています．当初採用されたPC/AT互換機での普及はもちろんのこと，PowerMacやRISCプロセッサ搭載サーバ/ワークステーションなど，ほかのアーキテクチャのコンピュータでも採用されています．

また，産業機器でVMEバスに代わって普及してきているCompactPCIバスも，電気的な規格はPCIそのものであり，産業機器向けに信頼性を上げて使用されています．とくにこの業界では10年や20年という製品保証期間が必要となるため，新しいバス規格が登場したからといって，すぐにバス規格を「鞍替え」するわけにはいきません．これは，VMEバスや，計測機器分野で使われるVXIバスが今日でもまだ利用されている状況からもあきらかです．

さらに最近では，組み込み機器向けCPUにPCIコントローラが内蔵されはじめ，情報家電機器の内部バスとして，PCIバスが採用されているものも珍しくありません．

● PCI-Xが登場した！

PCI-XはPCIをベースに，よりバースト転送を重視するプロトコルが採用されました．また各信号線の制御ルールを改定することで，より高速なクロックで動作させることもできるようになりました．これにより従来のPCIの2倍～4倍といった転送レートを確保できます．

さらにPCI-X 2.0では，従来のPCI-XをMode1とし，新たにDDRやQDR動作によるデータ転送レートを高めたMode2が定義されました．これにより後述のPCI-Expressに対抗し得るようなバス帯域を実現しています．

表1に現在までのPCIおよびPCI-Xの各モードと転送レートを示します．このように，PCIはPCI-X/Mode1，Mode2と進化をしつつ，性能の向上が図られています．

〔表1〕PCIおよびPCI-Xの各モードと転送レート

規　格	バス幅	クロック周波数	最大データ転送速度
PCI 2.3	32ビット	33MHz	133Mバイト/秒
PCI 2.3	32ビット	66MHz	266Mバイト/秒
PCI 2.3	64ビット	33MHz	266Mバイト/秒
PCI 2.3	64ビット	66MHz	533Mバイト/秒
PCI-X/Mode1	32ビット	66MHz	266Mバイト/秒
PCI-X/Mode1	64ビット	66MHz	533Mバイト/秒
PCI-X/Mode1	32ビット	100MHz	400Mバイト/秒
PCI-X/Mode1	64ビット	100MHz	800Mバイト/秒
PCI-X/Mode1	32ビット	133MHz	533Mバイト/秒
PCI-X/Mode1	64ビット	133MHz	1,066Mバイト/秒
PCI-X/Mode2-266	64ビット	133MHz	2,133Mバイト/秒
PCI-X/Mode2-533	64ビット	133MHz	4,266Mバイト/秒
PCI-X/Mode2-1066	64ビット	133MHz	8,532Mバイト/秒

PCI-X 1.0の仕様書が策定されたのは，1999年の秋となっています．それから現在までに4年が経過しましたが，現在のところIntel，AMD，ServerWorksといったベンダから，PCI-X対応ホストチップセットが登場し，サーバ/ワークステーションなどに採用されています．また秋葉原などでもデュアルXeonやOpteronなどサーバ向けのCPUの単体販売と並んで，PCI-Xを搭載したサーバ向けのマザーボードが店頭販売されているなど，入手は容易です．

PCI-X対応アドインカードとしては，ギガビットEthernetカードやRAIDカードなどが店頭に並んでいます．

● さらなる次世代高速バス

昨今，やれPCI-Expressだ，やれHyperTransportだ，Rapid-IOだ……と，話題になっている次世代バス規格があります．これらの新しいバス規格に共通する特徴は，差動伝送，小振幅差動インターフェース，低消費電力，1チャネルあたり数Gビット級のデータ転送能力など，どれを見ても現在求められているバスインターフェースの特徴/機能/性質を有しています．

これら三つのバスインターフェースは，LVDS（Low Voltage Differential Interface）という，350mVp-p（typ.）振幅の小振幅差動インターフェースが採用されており，従来のTTL/CMOSまたはLVTTL/LVCMOSとはまったく異なる信号です．伝送される情報のプロトコルの差異はあれ，基本的にはシリアルバスと呼ばれます．

● 差動シリアルバス規格の問題点

性能を追求するのであれば，将来的には確実にPCI-ExpressやHyperTransport，Rapid-IOに向かっていくことでしょう．しかし，それらの普及には時間がかかるのも事実です．

ここでPCI-Expressを例にとってみましょう．PCI-ExpressはPCIと比較すると，

- カードエッジ形状/コネクタ形状が違うために物理的互換性がない
- 信号の電気的互換性がない
- PCI-Expressの標準データ転送レートである1.25Gビット/秒に対応したデータ転送チャネルをもつデバイスの選択肢が少ない/あっても非常に高価
- PCIをはるかにしのぐ転送能力を得るには，結局は多ビットバス化が必要
- PCI-Expressバス規格を満たすデバイスの開発環境が高価

などという点があげられます．そして最大最強の問題（?）は，

- （少なくとも原稿執筆時点では）PCI-Expressを搭載したシステムがまだ存在しない

ということでしょうか．

● 従来資産の継承

もちろんデバイスの問題や開発環境の問題，そしてPCI-Expressを搭載したシステム（マザーボード）が存在しないという問題は，時間の経過とともに解決されるものです．しかし，物理的形状や電気的な部分の互換性は，時間が経てば解決するといった問題ではありません．

〔図1〕PCI-X/Mode1とMode2の間には川が，PCI-Expressの間には断崖絶壁の谷が!?



今回の特集で解説するPCI-X/Mode1は，物理的なカード/コネクタ形状はもちろん，電気的な部分でも従来のPCIとの互換性が考慮されています．だからこそ，第3章で解説するような，PCIバスを想定して設計されている評価ボードを，最低限の改造でPCI-X上で動作させるということも可能になるのです．

潤沢な資金と豊富な設計経験があり，長年使ってきたPCIバスシステムを捨て去る気迫で，新規開発案件としてPCI-Expressを採用するのであれば，筆者はむしろ応援します．しかし，既存システムの流用や製品寿命のことを考え，現在のPCIカードエッジバス，またCompactPCIバスシステムをそのまま活かして製品を開発しようとする場合には，筆者はPCIやPCI-Xのほうをお勧めします．

● PCI-XのMode1かMode2か

ひと口にPCI-Xといっても，大きく分けてMode1とMode2があります．もちろんMode2でも物理的な仕様はMode1と同一なのですが，じつはMode2は電気的特性の上でMode1と互換性がありません．PCI-X/Mode2ではV_{CC}電圧やI/O電圧が3.3Vおよび1.8VとなってしまったためPCIやPCI-X/Mode1とはI/Oインターフェースが別物になってしまったのです．

したがって，従来との互換性を重視するのであれば，PCI-X/Mode1が最適解ではないかと筆者は考えます．そうでなければ，少しだけがんばってPCI-X/Mode2を採用するか，もっとがんばってPCI-Expressなどの次世代シリアルバスを採用するしかありません(図1)．

● おまたせしました，PCI-Xです!

筆者は1997年からずっとPCIバスシステムを採用した製品設計に携わっています．その間，本誌および本誌増刊などに多数のPCIバスがらみの記事を執筆してきましたが，昨今ではFPGAも十分に高速に動くようになってきているので，もはやPCI-Xは難しいものではなくなったと宣言したいところです．

本特集が，PCIバスでデータ転送性能の上限に悩み，その解決策としてPCI-Xを模索/検討している読者の方の一つのバイブルになることができれば幸いです．

いくら・まさみ　来栖川電工有限会社

Column

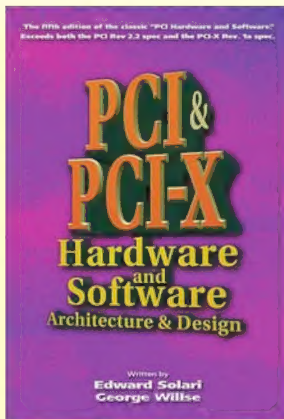
PCI-X 解説参考書

現在のところ，PCI-Xについて詳しく解説してある書籍としては，英語版では2冊ほどあるようです．

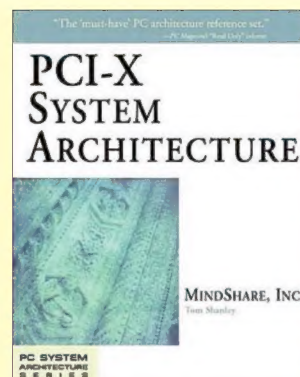
一つは，PCIのころから筆者もお世話になった通称「赤本」と呼ばれる『PCI Hardware and Software』が，PCI-Xまでを含んだ新版『PCI and PCI-X Hardware and Software: Architecture and Design』(写真A)になりました．日本語訳版はまだ予定はないようですが，要望が多ければ早期に実現するのではないでしょうか．

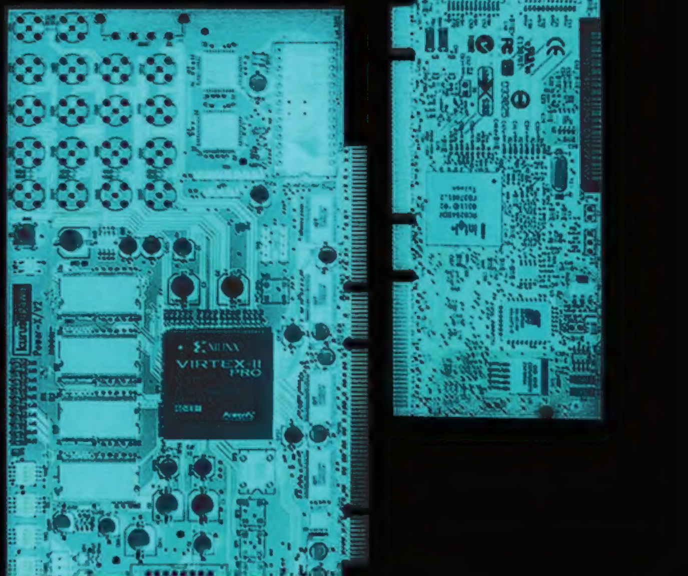
またもう一冊は，『PCI-X System Architecture(PC System Architecture Series)』(写真B)で，人によってはこちらのほうがわかりやすいという話も聞きます．

〔写真A〕PCI and PCI-X Hardware and Software: Architecture and Design, Edward Solari, George Willse(著)



〔写真B〕PCI-X System Architecture(PC System Architecture Series), Tom Shanley(著)





MPXバスの動作/シングル転送とバースト転送/バスプロトコルのいろいろ

バスとは何か――PCIの基礎知識

井倉 将実

PCI-Xは，PCIをベースにプロトコルの拡張などが施されたバスである．したがってPCI-X対応デバイスを設計するには，まずPCIについても理解しておく必要がある．ここではまず，PCIバスの基礎知識について解説する．（編集部）

1 PCIバスの基本的概念

● アドレスバスとデータバスの構成

図1（a）に一般的なCPUのバスを示します．アドレスバス，データバス，メモリ空間かI/O空間のどちらを選択したかを示す信号，そして読み出しか書き込みかを示す信号などから構成されています．このバスに接続する周辺デバイスは，CPUがアクセスしているのはメモリかI/Oか，そしてそのアドレスはどこかをアドレスデコーダで調べ，自分が選択されていれば，読み出し/書き込み信号の方向にしたがって，データバスの内容を読み書きします．

図1（b）にアドレスバスとデータバスがマルチプレクスされたバスの場合を示します．アドレスバスとデータバスが同一の信号になった代わりに，現在出力されているのがアドレスなのかデータなのかを示す制御線が新たに追加されています．またADバスの値はアドレスからデータに切り替わるので，アドレスデコーダにはアドレスを保持するレジスタを実装する必要があります．

● アドレスデコード回路をプログラマブルに

図1（a）も図1（b）も，アドレスデコーダは固定した回路になっているので，このままではプラグ＆プレイは実現できません．そこでアドレスデコーダをプログラマブルにするために，アドレス設定レジスタを用意して，アドレスデコード回路はアドレス保持レジスタとアドレス設定レジスタの内容を比較する回路を構成します．

この構成の場合，CPUはデバイスにアクセスする前に，各デバイスのアドレス設定レジスタに適切なアドレスを割り当てておかなければなりません．これをアドレス設定サイクルと呼ぶことにします．またこのときに使う信号は，それぞれのデバイスに専用の信号線が割り当てられている点に注目してください．これにより，まだアドレスが割り当てられていない状態でも，アドレス設定レジスタだけはデバイスごとに独立してアクセスが可能になるのです．

そして各デバイスにアドレスを割り当てた後，実際にバスアクセスを行うことができます〔図1（c）〕．

この図1（c）には，PCIバスがISAバスなどと異なるさまざまな要素が凝縮されています．

● PCIコンフィグレーションの概念

PCIバスを理解するときに，コンフィグレーション空間というものがよく理解できないという方がいます．たしかに，メモリやI/Oでもない第3の空間といわれても，抽象的でよくわからないでしょう．しかし難しく考える必要はありません．図1（c）でいえば$\overline{S0}$や$\overline{S1}$を使ってアクセスするレジスタの空間が，コンフィグレーション空間なのです．

● PCIのバスコマンド

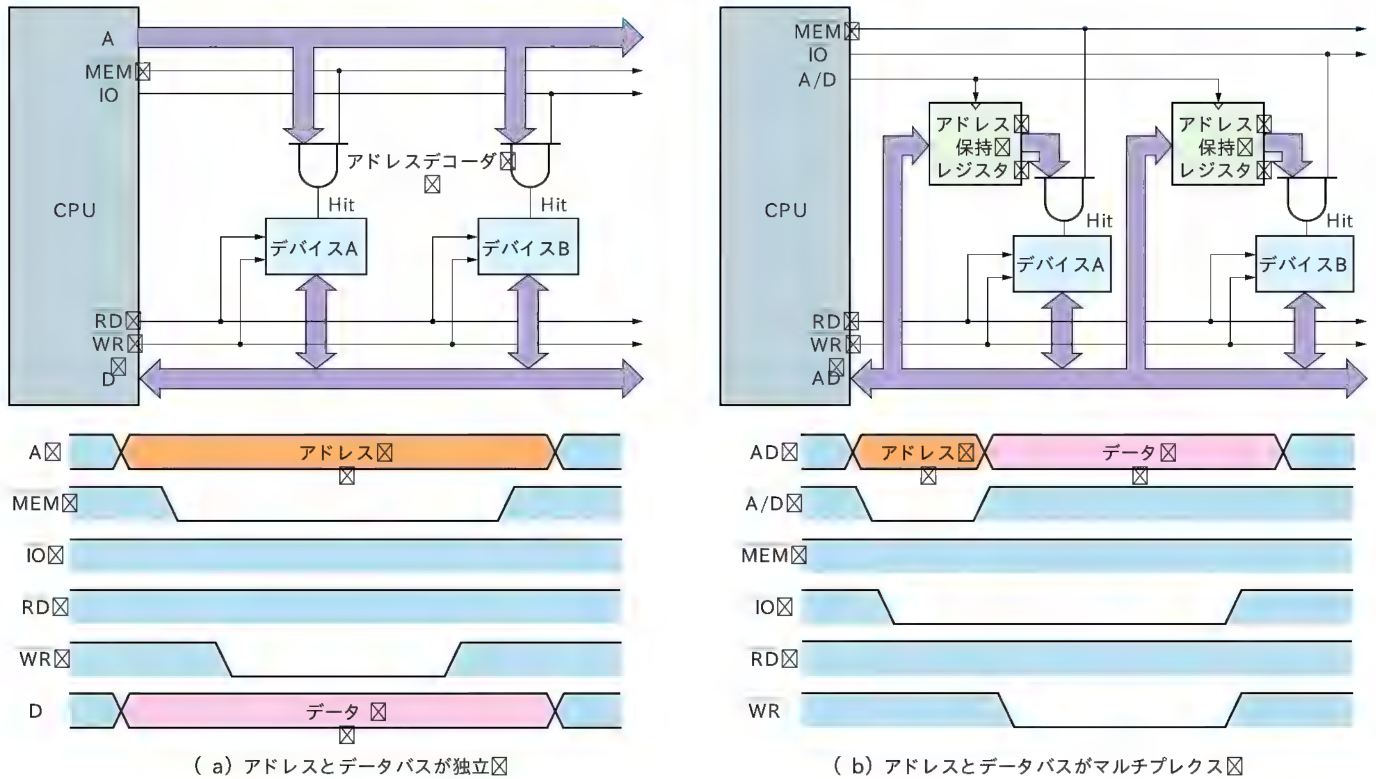
図1（c）を見るとわかりますが，$\overline{S0}/\overline{S1}$，$MEM/\overline{IO}$，$RD/\overline{WR}$の六つの信号線は，同時に複数の信号が“L”レベルになるタイミングはありません．これは情報が冗長になっていることを意味しています．そこでPCIでは，メモリなのかI/Oなのかコンフィグレーション空間なのか，そして読み出しなのか書き込みなのかを，4ビットにエンコードして表しています．これをバスコマンドと呼びます．

● データ転送の概念

データ転送を成立させるためには，まずデータ転送を行いたいデバイス（ここでは仮に親と呼ぶ）が「これからデータ転送をするぞ！」という宣言を出さなければなりません．またこの宣言と同時にアドレスとバスコマンドも出力します．バスにつながっている各デバイスは，このアドレスとバスコマンドをデコードして，転送相手が自分かどうかを確認します．もし自分が選択された場合は「はい，はい！そのアドレスは私です！」と手をあげなければなりません（ここではかりに子と呼ぶ）．

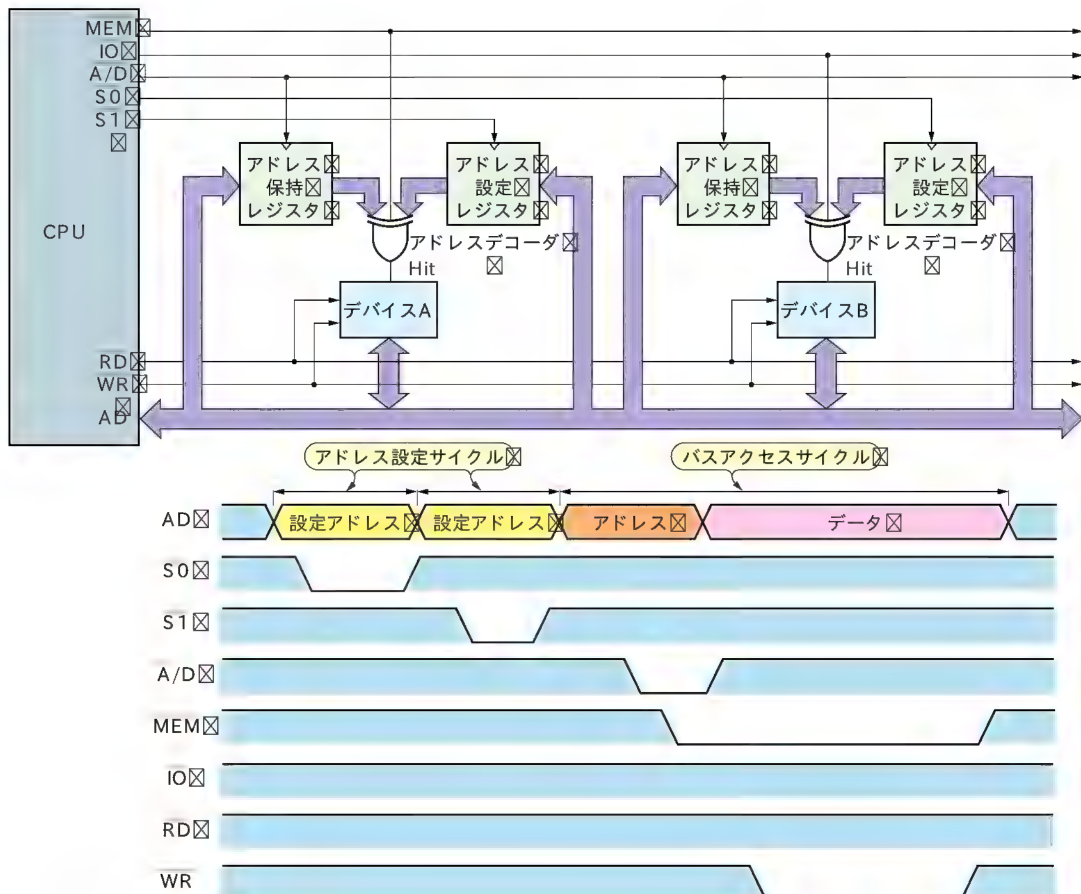
通常のバスであれば，この時点でデータ転送が成立すると考えられます．しかしPCIでは，この状態はデータ転送を行う親

〔図1〕アドレスバスとデータバス



（a）アドレスとデータバスが独立

（b）アドレスとデータバスがマルチプレクス



注：信号名の上にバーが付く
信号は負論理信号

（c）アドレスコードをプログラマブル化

と子の関係が決まっただけで，データ転送そのものは行われません．実際のデータ転送は，この後にもう1組の情報のやり取りが必要です．

たとえば親が子に対して書き込み動作をする場合，親は「書き込みデータの準備が完了しました．いまバス上に出力されている値は書き込みデータです（バスがマルチプレクスされているため）」と言わなければなりません．また子も「書き込みデータの受け取り準備は完了しました」と言う必要があります．お互いが「準備完了」を示した時点でデータ転送が成立します（図2）．

● バースト転送

バスがマルチプレクスされていると，同じバスをアドレスバスとデータバスで切り替えるため，パフォーマンスが上げられないように思われます．しかしPCIでは，連続したアドレスに対してのデータ転送の場合は，最初にアドレスを出力すれば，その後は自動的にインクリメントしたアドレスに対してのアクセスであると考え，データだけを次々転送するバースト転送という考え方が採用されています．PCIバスで転送レートをかせぐには，バースト転送が必須です．

なおPCIでは，複数ワードのデータを次々と転送するバースト転送に対して，データを1ワードしか転送しない動作をシングル転送と呼びます．

2 PCIバスの構成と概要

以降では，もう少し詳しくPCIバスの基本について解説します．

● 一般的なPCIバス搭載システムのバス構成

図3に一般的なPCIバス搭載システムのバス構成を示します．ホストCPUの下にはホストバスがあり，ホスト-PCIブリッジを介してPCIバスがつながります．ホスト-PCIブリッジは，通常一つのPCIシステムには1個しか存在しないものであり，CPUからのPCIバスへのアクセスを処理するための機能をもっています．

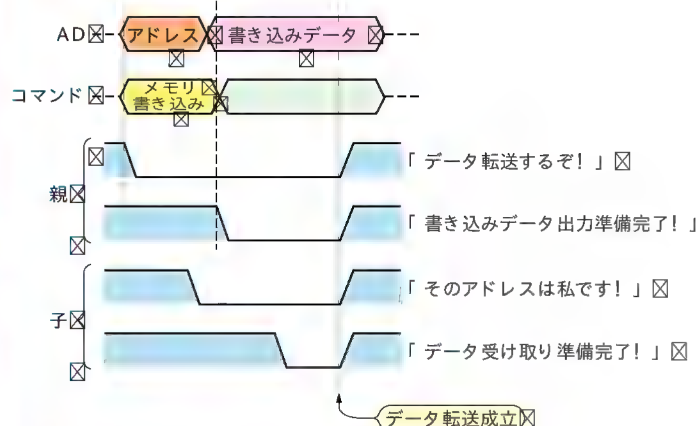
PCIバス上には複数のデバイスが存在します．さらにほかのバスへのブリッジが接続されることもあります．

● イニシエータ/ターゲット/エージェント

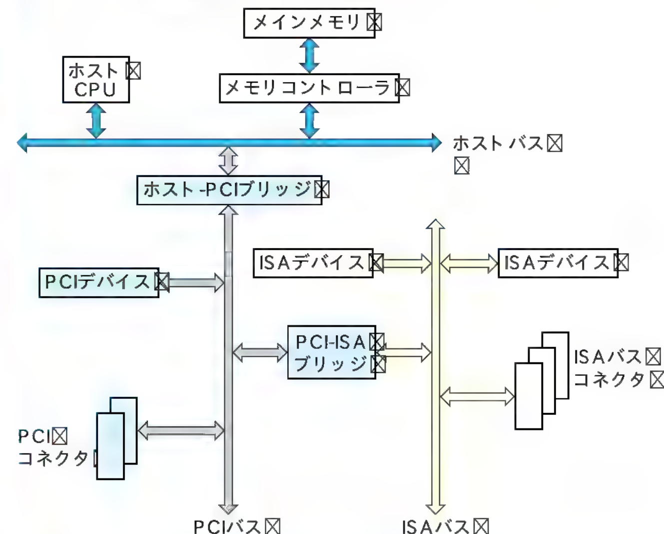
PCIバスでは，バスの制御権を要求しデータ転送を行うデバイスをイニシエータ，データ転送を受け付けるデバイスをターゲット，そしてバスの使用権を制御するバスアービタが存在します．またPCIバス上で行われるデータ転送をトランザクションと呼びます．そのトランザクションにかかわるイニシエータとターゲットをまとめてエージェントと呼びます．

さらにイニシエータとなってデータ転送を自ら行うデバイスを一般的にバスマスタデバイス，自分からはバスの制御権を取得せずデータ転送を受け付けるだけのデバイスをターゲットデバイスと呼びます（図4）．

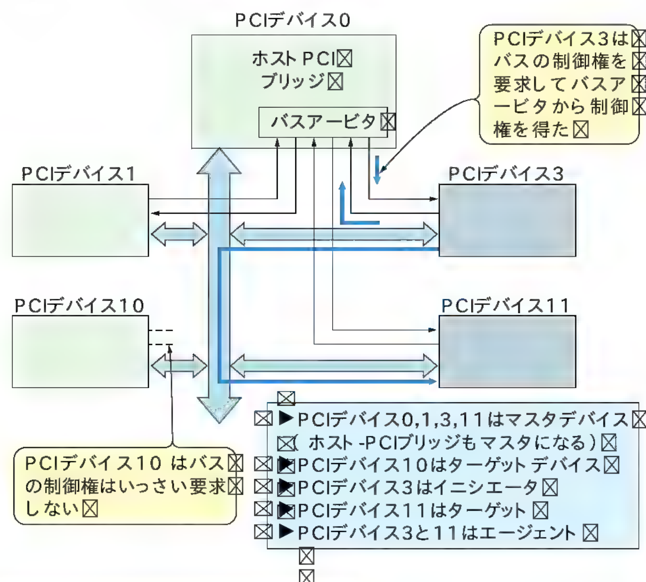
〔図2〕データ転送の概念



〔図3〕PCIバス搭載システムのバス構成



〔図4〕デバイスとエージェントの関係



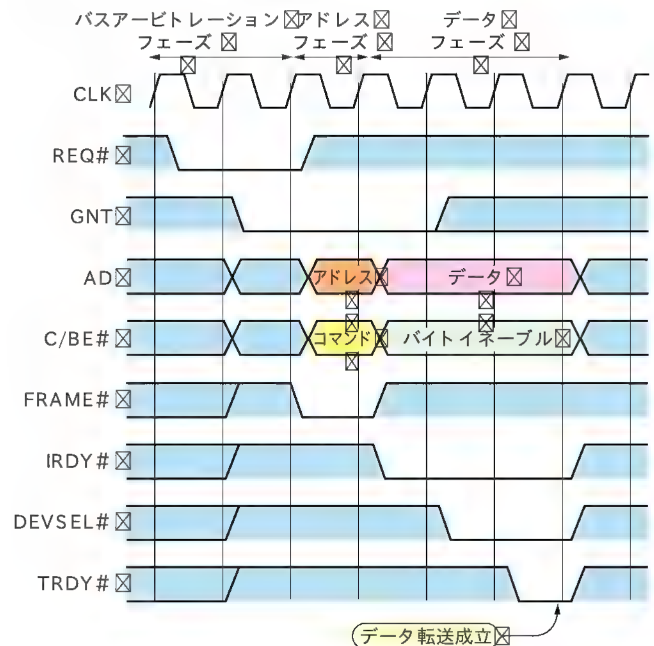
●トランザクションの構成

図5に実際のデータ転送のようすを示します．PCIでは，負論理の信号には信号名の最後に“#”が付きます．イニシエータは，まずバスアービタに対してバスの制御権を要求（REQ#を“L”レベル：アサート）します．GNT#がアサートされればバスの制御権を取得できたことになります．しかし，制御権を取得したタイミングでは，直前にバスを使っていたデバイスの転送がまだ終了していないこともあるので，バスがアイドル状態（FRAME#とIRDY#が“H”レベル：ディアサート）になるのを待ちます．

バスの制御権を取得した状態でバスアイドルを確認したら，いよいよトランザクションの開始です．FRAME#をアサートすると同時に，ADバスにアドレスを，C/BE#にコマンドを出力します．

選択されたターゲットはDEVSEL#をアサートしてDEVSEL応答を返します．そしてIRDY#およびTRDY#がどちらもアサート状態のとき，データ転送が成立します．

イニシエータがバスの制御権を要求してからバスサイクルを開始するまでの間をバスアービトレーションフェーズ，アドレスやバスコマンドを出力している期間をアドレスフェーズ，IRDY#やTRDY#がアサートされデータ転送が成立するまでをデータフェーズと呼びます．



〔図5〕トランザクションの構成

3 具体的なトランザクションの動作

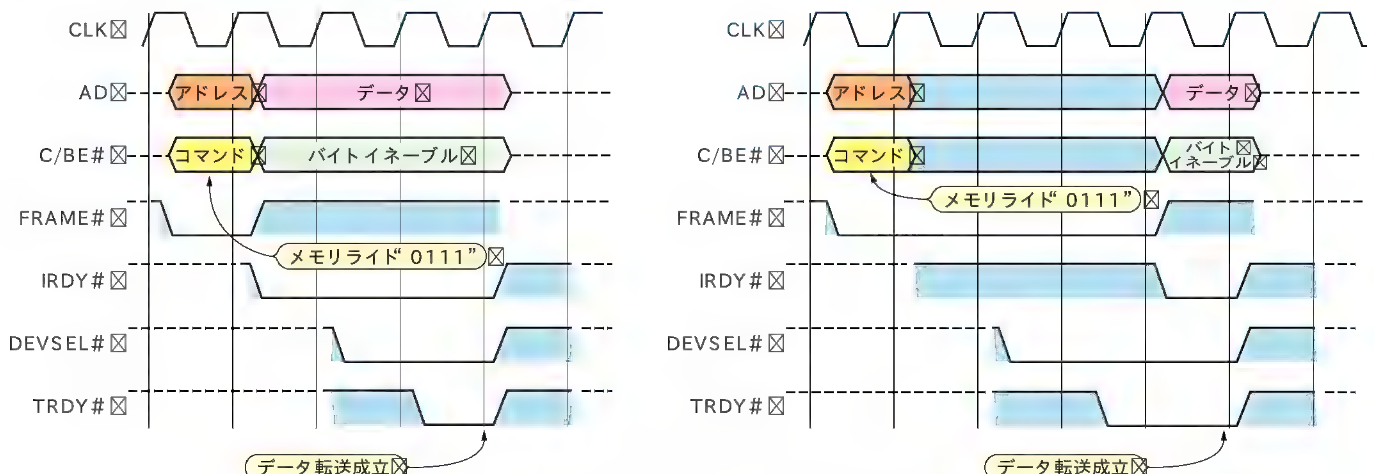
●シングル転送アクセスの動作

図6に，PCIバストランザクションでもっとも基本的な，シングル転送のメモリライトサイクルを示します．

トランザクションはFRAME#のアサートから始まるアドレスフェーズからスタートします．イニシエータはFRAME#をアサートすると同時にADバスにアドレスを，C/BE#にバスコマンドを出力します．この例ではメモリライトサイクルなので，C/BE#には“0111”が出力されています．

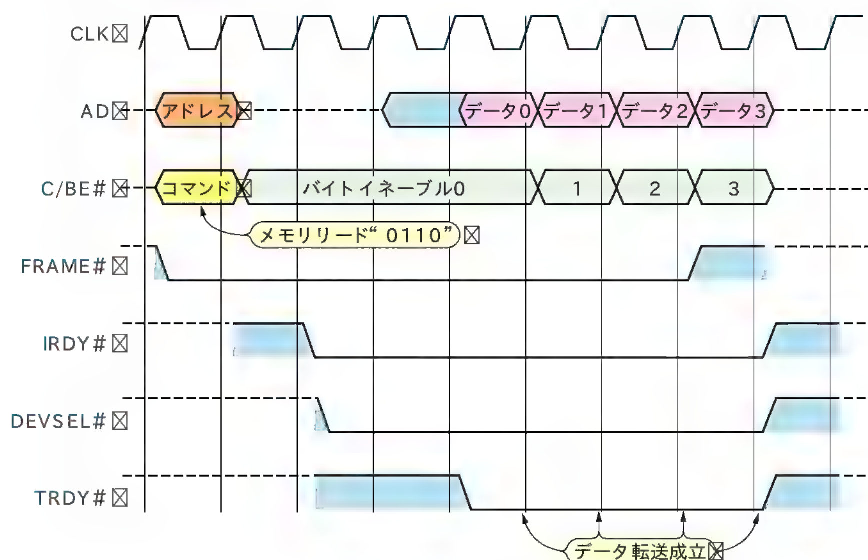
続いて，書き込みデータをADバスに出力し，C/BE#にバイトイネーブルを出力します．これでイニシエータ側の書き込みデータの準備が完了したのでIRDY#をアサートします．シングル転送で特徴的なのは，このとき同時にFRAME#をディアサートする点です．

アドレスフェーズはFRAME#をアサートした1クロック目のみです．しかしチップセットによっては，アドレスの次にすぐに書き込みデータを出力できないためか，2クロック以上FRAME#がアサート/IRDY#ディアサートされるものもあり

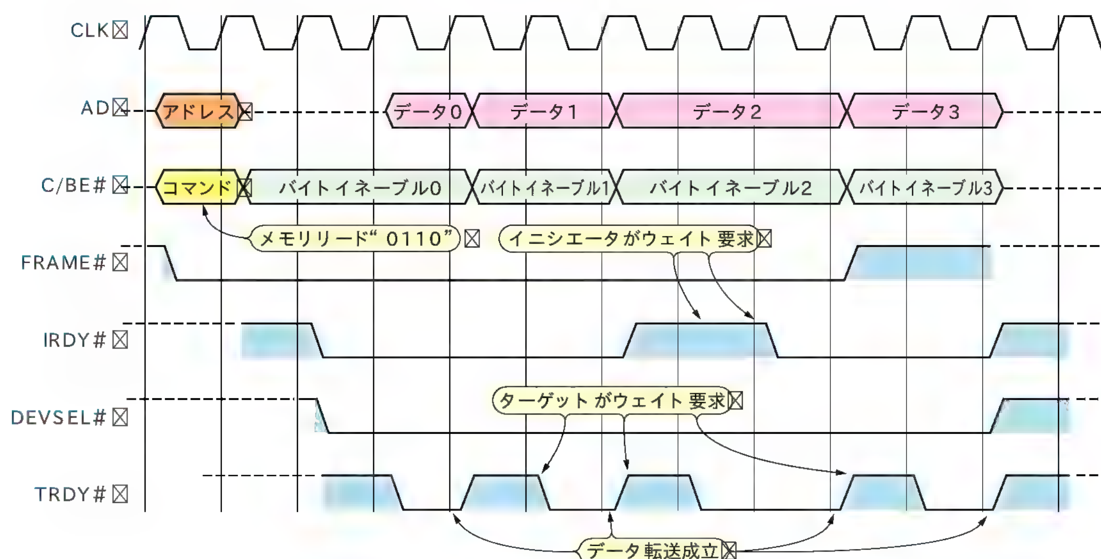


〔図6〕シングル転送アクセスの詳細

〔図7〕バースト転送アクセスの詳細



（a）2ワード目以後ノーウェイトで転送される例



（b）ターゲットが1ワード転送ごとにウェイトを入れる例

ます〔図6（b）〕．この期間はアドレスフェーズではなく，書き込みデータの準備ができていない，イニシエータ側のウェイト状態を示します．

ターゲットはFRAME#のアサートを検出したら，そのときのADバス上の値をアドレスとして，C/BE#バス上の値をバスコマンドとして取り込み，自身に対するアクセスであるかどうかをデコードします．自分が選択されていることを認識したら，DEVSEL#をアサートします．そして，書き込みデータを受け取る準備が完了したら，TRDY#をアサートします．よって，DEVSEL応答と同時にデータを受け取る準備ができているのなら，DEVSEL#のアサートとTRDY#のアサートは同時でもかまいません．

ターゲットは自身のTRDY#のアサートとイニシエータのIRDY#のアサートとを認識すると，そのときのADバスの値を書き込みデータとして，そしてC/BE#バス上の値をバイトイネーブルとしてデバイス内部に取り込みます．さらにターゲットは，ターゲットのバックエンドの仕様にしたがい，取り込んだ書き込みデータを該当するレジスタやメモリに書き込みます．

データ転送が成立した次のクロックでは，イニシエータはIRDY#をディアサートします．シングル転送なのでFRAME#はすでにディアサート状態ですが，ここでFRAME#の信号ド

ライブを開放(ハイインピーダンス状態に)します．以降はマザーボード側のプルアップ抵抗でディアサート状態が保たれます．また，ADバスやC/BE#のドライブも開放します．ターゲットはTRDY#およびDEVSEL#を同時にディアサートします．

そして次のクロックで，イニシエータはIRDY#のドライブを開放します．ターゲットはDEVSEL#とTRDY#のドライブを開放して，一連のトランザクションは終了します．

● バースト転送アクセスの動作

先ほどは1ワードのデータ転送を行うトランザクションを説明しましたが，次はマルチプレクスされたPCIバスで最高のパフォーマンスでデータ転送を行うためのバースト転送について説明します．

図7にバースト転送でのメモリリードサイクルを示します．バースト転送におけるイニシエータの動作は，アドレスフェーズまではシングル転送と同一の動作となりますが，データフェーズ中でもFRAME#がアサートされ続ける点が大きく異なります．

またリードサイクルではイニシエータが読み出し要求をしているので，アドレスフェーズの次のデータフェーズに切り替わる時点でADバスが開放され，DEVSEL応答を返したターゲットにADバスのドライブをまかせます．

ターゲットは，DEVSEL応答を返してから最初のデータ転送までは，バースト転送でもシングル転送でも同じ動作です．しかし，FRAME#と同時にIRDY#もアサートされていることを確認することで，イニシエータがバースト転送を要求していることを判定し，さらに次のアドレスの読み出しデータを準備して，ADバスに出力します．ターゲット側の仕様でノーウェイトでリードデータを用意できない場合は，TRDY#をディアサートしてウェイト状態を示すことができます〔図7(b)〕．

イニシエータは，あと1ワードのデータを転送すればバースト転送を終了させるというタイミングで，FRAME#をディアサートします．これによりターゲットに，次でデータ転送が成立すれば最後のデータ転送であることを示します．つまり，最終ワードのデータ転送は，シングル転送アクセスのバスの動作と同様になるわけです．

最終ワードの転送が成立したら，シングル転送のときと同様に，イニシエータはIRDY#をディアサートし，FRAME#やC/BE#のドライブを開放ます．ターゲットはDEVSEL#とTRDY#をディアサートします．そして次のクロックで，イニシエータはIRDY#のドライブを開放します．ターゲットはDEVSEL#とTRDY#のドライブを開放して，一連のトランザクションは終了します．

● 転送レートを計算する

クロック周波数33MHzの32ビットという一般的なPCIバスでは，シングル転送で1ワードのデータを転送するのに4クロックかかるとすると，最高でも33Mバイト/秒の転送レートになります．

バースト転送で，アドレスフェーズから最初のデータ転送準備まで4クロックかかり，その後のデータ転送をノーウェイトで8ワード転送した場合には，8ワードあたり12クロックの時間で転送できるので，最高89Mバイト/秒の転送レートになります．これにより，アドレスフェーズ分のオーバヘッドを減らすことができ，バスの使用効率が向上します．PCIバスでシステムを効率よく動かすためには，バースト転送を実装することが必要となるわけです．

〔表1〕PCIのバスコマンド一覧

バスの動作			C/BE[3:0]#
メモリサイクル	リード	メモリリード	“0”“1”“1”“0”
		メモリリードライン	“1”“1”“1”“0”
		メモリリードマルチプル	“1”“1”“0”“0”
	ライト	メモリライト	“0”“1”“1”“1”
		メモリライト＆インバリデート	“1”“1”“1”“1”
I/Oサイクル		I/Oリード	“0”“0”“1”“0”
		I/Oライト	“0”“0”“1”“1”
コンフィグレーションサイクル	コンフィグレーションリード		“1”“0”“1”“0”
	コンフィグレーションライト		“1”“0”“1”“1”
インタラプトアクノリッジサイクル			“0”“0”“0”“0”
スペシャルサイクル			“0”“0”“0”“1”
デュアルアドレスサイクル			“1”“1”“0”“1”
予　約			“0”“1”“0”“0”
			“0”“1”“0”“1”
			“1”“0”“0”“0”
			“1”“0”“0”“1”

● PCIバスコマンド

表1に，PCIのバスコマンド一覧を示します．アクセス先の空間別に，メモリアクセス系コマンド，I/Oアクセス系コマンド，コンフィグレーションアクセス系コマンドにわけられます．表を良く見ると，メモリアクセス系コマンドには単なるメモリリードやメモリライト以外にもリード/ライトコマンドがあります．これはPCIメモリ空間をキャッシュする場合に使われるバースト転送専用のコマンドです．

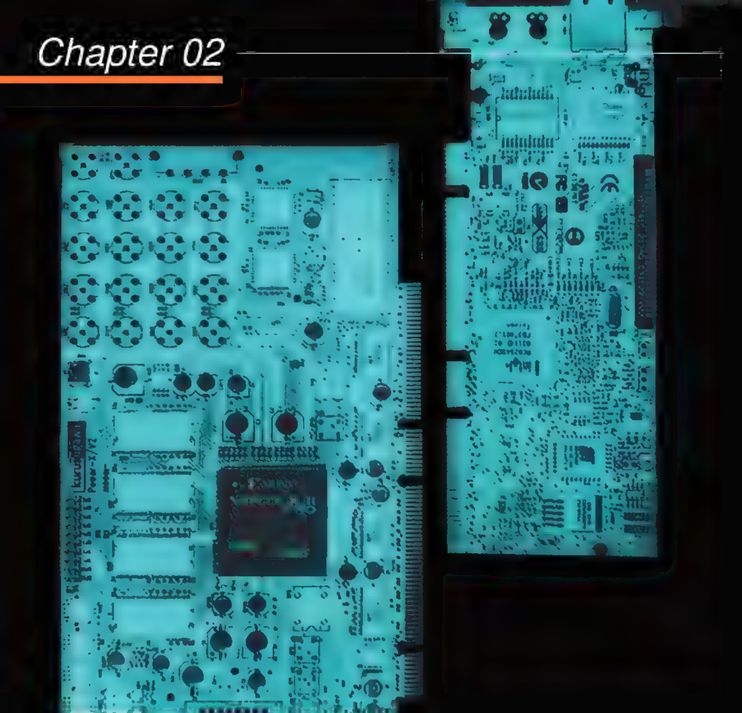
それ以外のインタラプトアクノリッジサイクルやスペシャルサイクル，デュアルアドレスサイクルは特殊な用途のコマンドであると考えて間違いありません．

● トランザクションターミネーション

トランザクションは，必要な転送をすべて正常に終えて完了するほかにも，いくつかの終了方法(ターミネーション)があります．

イニシエータがアクセスしたアドレスにターゲットデバイスが存在しなかった場合は，マスタアボートという終了になります．またターゲット側で何らかの都合でトランザクションを終了させたい場合は，リトライ，ターゲットアボート，ディスコネクトの3種類のターミネーションが規定されています．

いくら・まさみ　来栖川電工有限会社



PCIとの違いから
スプリットトランザクションの動作まで

PCI-Xバスプロトコルの詳細

井倉 將実

PCI-Xを理解するためには，新しく追加されたアトリビュートフェーズと，従来のPCIと比較したバスコマンドやトランザクションターミネーションルールの違い，そしてスプリットトランザクションの動作を把握する必要がある．ここでは従来のPCIとPCI-Xを比較しながら，PCI-Xで変更されたバスコマンドや新規に規定されたプロトコルなど，PCI-Xバスプロトコルについて解説する．（編集部）

はじめに

現在のところ，PCI-Xの最新バージョンは2.0aです．PCI-X2.0からは従来のPCI-X1.0の動作をMode1，そして1クロックで複数のデータを送るDDR（デュアルデータレート）転送や，さらにQDR（クワッドデータレート）転送に対応したMode2が規定されました．プロローグでもふれているように，PCI-XのMode1とMode2は電気的にもかなり異なります．

本特集では，従来のPCI（コンベンショナルPCIとも呼ぶ）と電気的に互換性のあるPCI-X/Mode1を対象に解説を進めます．

1　従来のPCIの問題点

PCI-XがPCIバス規格とどれだけ違っているのかを知る前に，まずPCIでデータ転送レートを向上させるために何が問題となっているか説明しましょう．

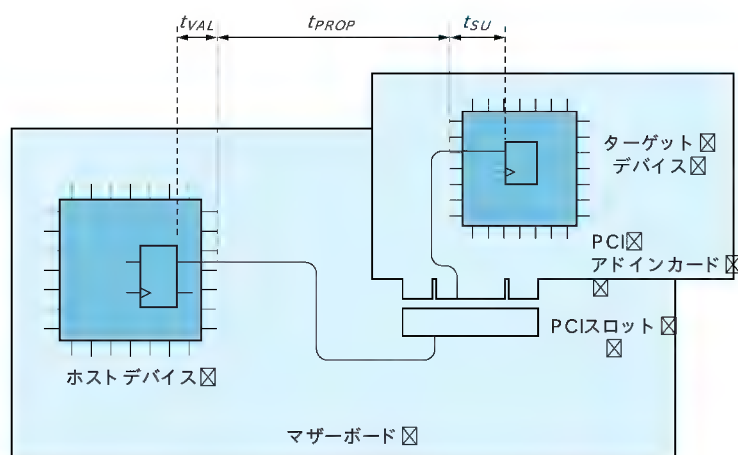
● バスクロックを上げにくい

PCIバスはバスインターフェースとはいえ，かなり複雑なステートマシンを組む必要があります．とくに，データフェーズ中のDEVSEL#/TRDY#/STOP#信号の状態によりターミネーションを判定して次に何をすべきかを考える必要があるイニシエータデバイスの設計の場合は，それがより顕著になります．さらに，連続したバースト転送をサポートしようとすると，FIFOやデュアルバッファリング技術などを多用する必要もあり，回路規模がかなり大きくなります．

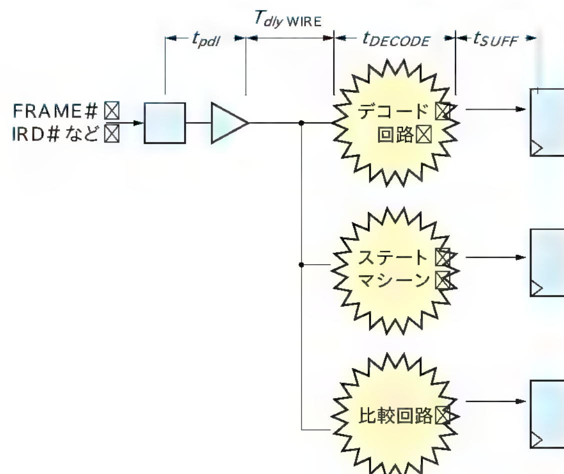
PCIのステートマシンは，クロックの立ち上がりに同期して動作し，その時点でのステートと入力信号の状態により出力信号や次のステートを決定します．そのため，ステートマシンの中でバスの信号ピンを直接参照しなければならず，ステートマシンの駆動クロックは，さまざまな要因で遅くなることになります．

図1（a）はPCIシステムでの信号遅延状態を表したものです．

〔図1〕PCIバスでの各種遅延時間



（a）信号遅延状態



（b）FPGA内の遅延

PCI クロックの立ち上がりからの出力パッドの遅延時間(t_{VAL})+基板上の配線遅延時間(t_{PROP})+入力レジスタのセットアップタイム(t_{SU})がシステムの最小動作周期となり，この逆数が最高動作周波数となります．

さらにデバイスがFPGA の場合，t_{SU} をもっと細かく見ると，入力パッドの遅延時間(t_{pdl})+IC内部の配線遅延時間($t_{dlyWIRE}$)+ステートマシンのデコード遅延(t_{DECODE})+フリップフロップのセットアップタイム(t_{SUFF})となります(図1(b))．

デバイスのピンレベルでの遅延は，外側の信号をドライブするデバイスの t_{VAL} や，基板上の配線遅延も加算されています．これがクロック 66MHz の PCI バスとなると，ワーストケースで t_{VAL} が 6ns，t_{PROP} が 5ns，そして t_{SU} が 3ns と要求されます(じつはさらに信号スキューの 1ns も含まれる)．FPGA の入力ピンからステートマシンを構築するフリップフロップのセットアップ時間がわずか 3ns というのは，非常に厳しい条件となります．

FPGA ではほとんどの場合，設計を繰り返すたびにデバイス内部における配置配線位置が変化するため，デバイス内部の配線遅延時間は変動します．さらに，ステートマシンのデコード遅延は，ステートマシンが複雑になったり，32ビットのレジスタを直接ステートマシン内で駆動すると，DEVSEL# 信号などがつながるレジスタ数が複雑になりファンアウトが増えます．1相クロックで設計する場合にはこの設計手法しかないでしょう．このため，さらに配線遅延時間が増えることになります．

組み込み系システムの場合には，t_{PROP} は限りなくゼロに近くすることはできますが(5ns を配線長に換算すると標準的な FR-4基板でおよそ 65cm)，デバイスのもつパラメータである t_{VAL} や t_{SU} は努力しようにも限界があります．

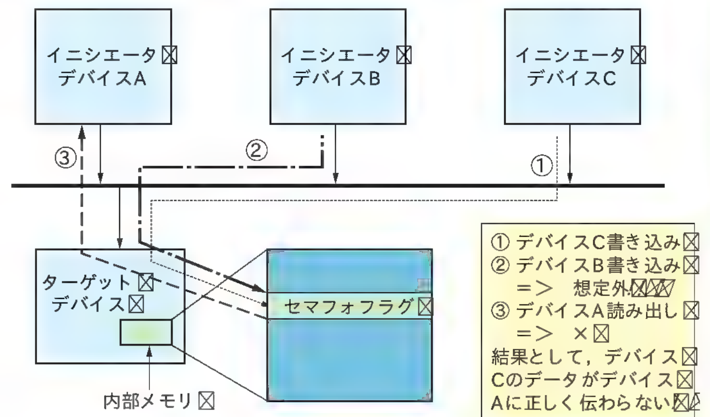
● どのバスマスタからアクセスされているかわからない

次は，バスプロトコル上の問題です．たとえば図2のように，一つのターゲットデバイスに対して複数のマスタデバイスからアクセスがある場合を考えます．

デバイスCが書いたステータス値をデバイスAが読み取り，別の値に書き戻した後に再度Cデバイスが読み出すというような処理を行っている最中に，デバイスBがバス制御権を取得しこの値を上書きしてしまう可能性もあります．これはセマフォの考えそのものです．PCI バスにはリードモディファイライトに相当するバスコマンドがないので，通常のリードアクセスとライトアクセスの二つに分けて発行する必要があるためです．

また，たとえばデバイスCのあるレジスタへのアクセスは時間がかかるので，最初にそのアドレスにアクセスがあった場合は1度リトライを返すデバイスだとします．このレジスタに対してデバイスAが最初にアクセスをはじめリトライが返ってきた後，デバイスBが同じデバイスCのそのレジスタにアクセスしてしまったらどうなるでしょうか．本来はデバイスAのために用意したデータを後からアクセスしてきたデバイスBに対して返してしまい，リトライサイクルにより再度バスアクセスを発行したデバイスAが，またもリトライを返されてしまうという事態が発生します．

〔図2〕どのバスマスタからアクセスされているかわからない



このように，本来なら一度のバストランザクションで終了させたいアクセスが，PCI バス上では複数のバストランザクションに分けなければならないとき，正しいタイミングでアクセスを行うには，何らかの方法で，アクセスの排他制御を行わなければならないことになります．

じつは PCI バスの規格には，排他アクセスを行うための LOCK# 信号があります．この信号はホスト-PCI ブリッジもしくは PCI-PCI ブリッジ，そして拡張バスブリッジなどの間での排他的データ転送処理に使われるものであり，一般のデバイス間のデータ転送に使用するものではありません(PCI-Specification Appendix-F 参照)．PCI-PCI ブリッジのように，プライマリバス側とセカンダリバス側のアクセスが完了していない間，ほかのイニシエータがアクセスすることを許可できないような場合には LOCK# 信号を使って排他処理ができるのですが，通常のデバイスからの LOCK# 信号は無視されます．

よって，従来の PCI バス規格では，残念ながら排他処理を確実に実現する方法はありません．

● 転送データ長がわからない

ターゲットからみると，イニシエータからバースト転送が要求されても，何ワードのデータが転送されるのかわからないため，ターゲットデバイス内に内蔵されたバッファや FIFO の空き状況をみて，受け付けるだけデータを受け付け，ある時点でバッファがあふれそうになるとディスコネクトやリトライ要求を行うことになります．

逆にイニシエータからターゲットを見ると，バースト転送を行ったとき，ターゲットはいったい何ワードのデータ転送に応答できるのかということがわかりません．このためイニシエータは，絶えずターゲットからの応答を監視し，1クロックごとにターミネーション要求がないかどうかを判定し，次のデータ転送を行わなければなりません．実際，リトライやディスコネクトが返された場合は，アドレス/データともに保持バッファを用意して，FIFO などから読み出したデータを捨てないよう

にしなければなりません．

イニシエータにしてみればあと何ワード分転送すべきか，ターゲットにしてみればあと何ワード分受け取らなければならないかがわかっていると，ハードウェア設計時におけるバッファサイズの決定や運用時のデータ転送能力（パフォーマンス）をある程度見きわめることができることでしょう．

たとえば，あるターゲットデバイスは128ワードのデータを連続して受け付けることができるとわかっていれば，このターゲットデバイスに対してのデータ転送では，128ワードまでをノーウェイトで転送することで，最高のパフォーマンスを発揮させることができるでしょう．

しかし従来のPCIでは，バースト転送開始時点で転送すべきデータ転送長がわからないために，ターゲットとイニシエータがお互いに「もういいかい？」，「ま～だだよ」というような掛け合いを，毎回行っているわけです．ターゲットのバッファがいっぱいになると突然「はいストップ！」といわれるわけですが，その方法にも，ターゲットがTRDY#をディアサートしてウェイト状態を要求する場合や，ウェイト時間が長い場合はディスコネクトされるかもしれません．しかもディスコネクトにはデータ転送を伴うディスコネクトとデータ転送を伴わないディスコネクトの2通りあります．これもイニシエータ回路が複雑になり，結果的に先に説明したシーケンサの複雑化による動作速度の向上が見込めないということにもつながっているわけです．

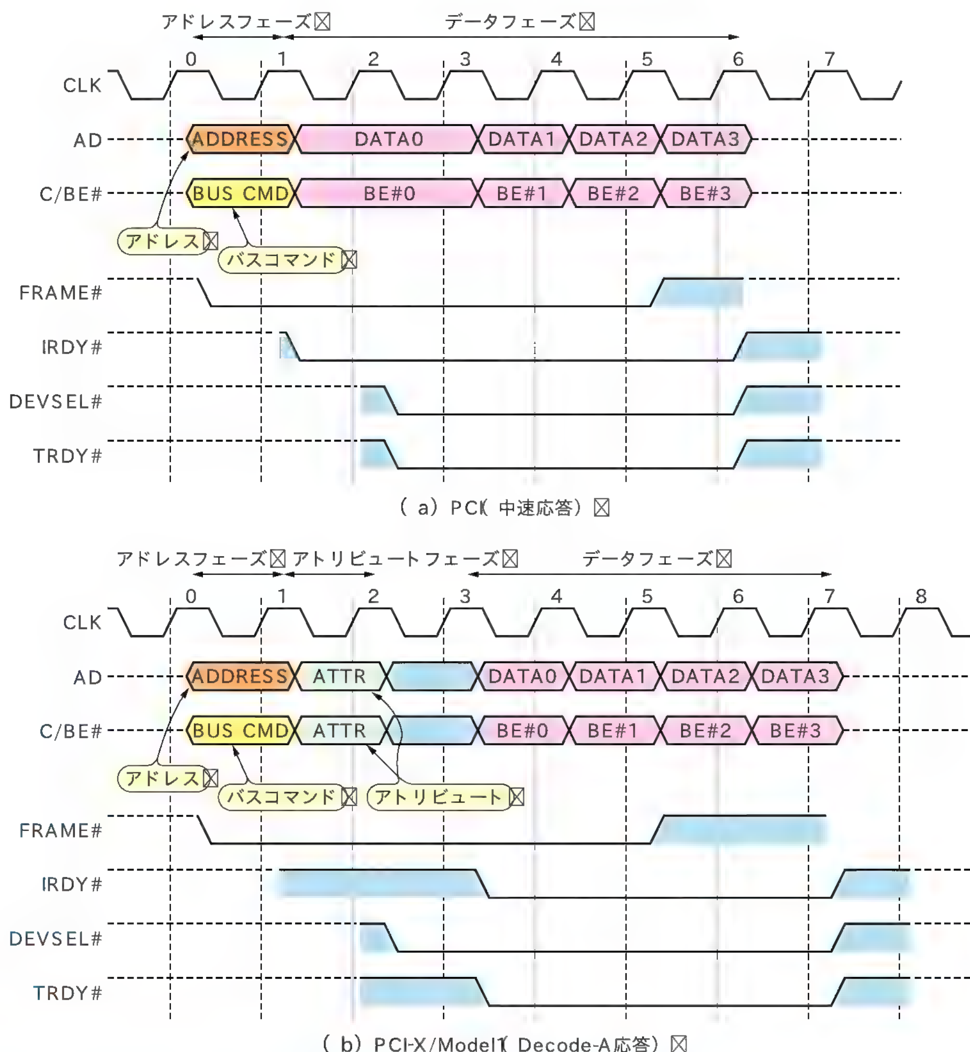
2 PCI-Xバストランザクションの概要

● リクエスタ/コンプリータ

従来のPCIでは，トランザクションを開始するデバイスをイニシエータ，そのトランザクションを受け付けるデバイスをターゲット，そしてトランザクション中の両者をエージェントと呼びました．

PCI-Xでは後述するスプリットトランザクションにより，トランザクションの開始とそれを受け付けるデバイスの立場が逆転する場合があるため，トランザクションを開始するデバイス

〔図3〕従来PCIとPCI-Xのデータ転送の比較



をイニシエータと呼ぶと，混乱をきたします．そこで，通常のバスサイクルを開始するデバイス（従来イニシエータと呼んでいたデバイス）を，データ転送を要求するデバイスとしてリクエスタと呼びます．それに対してバスアクセスを受け付けるデバイス（従来ターゲットと呼んでいたデバイス）を，データ転送を成立させるデバイスとしてコンプリータと呼びます．

スプリットトランザクション以外では，（慣れていることもあるので）従来どおりイニシエータとターゲットと呼ぶ場合もあります．

また，PCI-X の規格書では明示的に従来の PCI を指す場合，コンベンショナル PCI と呼んでいます．

● 従来 PCI と PCI-X のデータ転送の比較

PCI-X は PCI と同じ信号線を使うので，一見して PCI バスの延長のように見えるのですが，筆者の経験でいうと，中身はまったくといってよいほど別の動作をすると考えたほうが理解が早いかもしれません．

図３に従来 PCI と PCI-X の実際のデータ転送のバス動作を比較します．トランザクションとしてもっとも大きな違いは，アドレスフェーズの次にアトリビュートフェーズが追加された点でしょう．

以降では，PCI-X で追加されたアトリビュートや，新たに規定されたバスコマンド，変更された FRAME# の挙動やバイトイネーブル情報の取得方法などについて解説していきます．

3 アトリビュート

ここでは，アトリビュートフェーズで出力される情報のフォーマットなどについて説明します．

● どこの誰が発行したトランザクションか?/転送長は?

従来の PCI での問題点として，どこの誰が発行したトランザクションなのかわからないという点と，バースト転送開始時に転送データ長が不明である点を説明しました．PCI-X ではこれらの情報をアトリビュートフェーズに追加することで解消しています．PCI-X ではすべてのトランザクションのアドレスフェーズの直後に，必ずアトリビュートフェーズが存在します．

図４にアトリビュートフェーズのフォーマットを示します．バスコマンドの種類やシングル転送かバースト転送の違いによって，アトリビュートのフォーマットにもいくつか種類があります．もっともよく使われるのが，DWORD トランザクションリクエスタアトリビュート（シングル転送用）とバーストトランザクションアトリビュート（バースト転送用）の二つでしょう．

じつはこれ以外にもう一つ，コンフィグレーションコマンド時のアトリビュートの規定もあるのですが，それについては最後に解説します．

〔図4〕アトリビュートのフォーマット

C/BE[3:0]#	AD[31:00] 31	30	29	28 ～ 24	23 ～ 16	15 ～ 11	10 ～ 8	7 ～ 0
バイトイネーブル	R	NS	RO	タグ	リクエスタ：バス番号	デバイス番号	ファンクション番号	（未使用）

（a）DWORDトランザクションアトリビュート（シングル転送用）

C/BE[3:0]#	AD[31:00] 31	30	29	28 ～ 24	23 ～ 16	15 ～ 11	10 ～ 8	7 ～ 0
上位バイト転送数	R	NS	RO	タグ	リクエスタ：バス番号	デバイス番号	ファンクション番号	下位バイト転送数

（b）バーストトランザクションアトリビュート（バースト転送用）

C/BE[3:0]#	AD[31:00] 31	30	29	28 ～ 24	23 ～ 16	15 ～ 11	10 ～ 8	7 ～ 0
上位バイト転送数	BCM	SCE	SCM	（未使用）	リクエスタ：バス番号	デバイス番号	ファンクション番号	下位バイト転送数

（c）スプリットコンプリーションアトリビュート

C/BE[3:0]#	AD[31:00] 31	30	29	28 ～ 24	23 ～ 16	15 ～ 11	10 ～ 8	7 ～ 0
上位バイト転送数	IR	R	RO	（未使用）	リクエスタ：バス番号	デバイス番号	ファンクション番号	下位バイト転送数

（d）デバイスIDメッセージアトリビュート

● リクエスタバス番号/デバイス番号/ファンクション番号

トランザクションを発行したリクエスタのバス番号/デバイス番号/ファンクション番号が格納されます．この三つの値をまとめてリクエスタIDと呼びます．これらの値は，スプリットトランザクションや排他アクセス処理を行う場合などに使用されます．

従来のPCIでは，排他アクセス制御を行うのは事実上不可能でしたが，アトリビュートフェーズにより「どこの誰がアクセスを行っているのか」ということを知ることができるようになり，完全な排他アクセス制御を行えるようになりました．

また，スプリットトランザクションによるスプリットコンプリーションコマンドの場合は，リクエスタの情報ではなくコンプリータのバス番号/デバイス番号/ファンクション番号が格納されます〔図4(c)〕．

● データ転送長とバイトイネーブル

DWORDトランザクションアトリビュートでは，C/BE#バスにはバイトイネーブルが出力されます．リードサイクルの場合にはイニシエータが必要とするバイト位置の情報が，ライトサイクルの場合には書き込むべきデータバウンダリ情報がセットされます．

バーストトランザクションアトリビュートでは一部のコマンドを除きバイトイネーブル情報は不要です．データバスが32ビットなら4バイト，64ビットなら8バイトすべてのバイト位置が有効とみなされます．またバースト転送時は転送長がC/BE#とADのビット7〜0に出力されます．VHDLで記述すると，データ転送長はバイト単位で，

```
ByteCOUNT <= C_nBE(3 downto 0) &
                       PCIAD(7 downto 0) ;
```

となり，合計で12ビット長のデータ転送カウンタとなります．C/BE#の4ビットが上位，AD[7:0]が下位である点に注意してください．

データ転送長は1バイト単位で設定でき，001hの場合は1バイト，002hの場合は2バイトと，数値そのものがデータ転送バイト数を示します．よってFFFhは4095バイトで，最大転送長は000hの4096バイト(4K)となります．

イニシエータは，トランザクションを発行するたびにデータ転送長をアトリビュートフェーズに出力し，ターゲットからディスコネクトが発行されたら，そこまで行ったデータ転送バイト数分を減じて，次のトランザクションを発行します．

● そのほかの情報

このほか，シングル/バースト転送のいずれのアトリビュートにも共通しているものがいくつか割り当てられています．

▶ タグ(TAG)

トランザクションのIDです．5ビットの値であり，おもにスプリットトランザクション時に使用されます．

▶ リラックスオーダリング(RO)

トランザクションによっては，アクセスとアクセスの順番を入れ替えてもデータ転送が成立する場合があります．まったく別のアドレスへのアクセスがあるのであれば，リードとライトはその順序を入れ替えても問題はありません．CPUやMPUの内部ロジックでも，実行する命令の順番を入れ替えて効率よくパイプラインを駆動させるアウトオブオーダという手法が取り入れられていますが，PCI-Xのトランザクションでも同様なことが可能となります．

▶ ノースヌープ(NS)

キャッシュシステムを構築した場合に，続くデータフェーズのはじめから終わりまで，システムのキャッシュには含まれない＝非キャッシュ領域のデータであることを通知するビットです．

PCI-Xのバスは通常非キャッシュ領域におかれるため，メモリアクセスでは1にします．また，メモリ以外の空間に対するアクセスの場合は0にします．

4 バスコマンド

表1に，従来のPCIとPCI-Xのバスコマンドを示します．

● PCIと同じ名前のコマンド

従来のPCIと同じ名前のバスコマンドには，次のようなものがあります．

Column

PCIとPCI-Xのモード判定について

PCI-Xでは，アドインカードがPCI-Xに対応していることを示すためのピンが規定されています．PCI-Xのクロック周波数66MHzに対応している場合は，サイドBの38番ピンに10kΩのプルダウン抵抗を実装します．133MHzに対応している場合はNC(開放状態)にしてください．コンベンショナルPCIでは，このピンはGNDに接続されています．この信号ピンの状態を見ることで，システム側はPCI-X対応スロットにコンベンショナルPCIボードが実装されているか，66MHzのPCI-X対応ボードが実装されているか，133MHzのPCI-X対応ボードが実装されているかを判断できます．

逆にPCI-X対応デバイス側が，現在どのシステムで動いているかを判定することもできます．PCIバスのリセット解除時のTRDY#とSTOP#の状態によりそれを判定できます．どちらもディアサート状態ならばコンベンショナルPCIモード，TRDY#のみアサートされていればクロック66MHzのPCI-Xモード，どちらもアサートされていればクロック133MHzのPCI-Xモードとなります．またSTOP#のみアサートされている場合は，クロック100MHzのPCI-Xモードとなります．クロック100MHzのモードは，一つのPCIバス上にクロック133MHz対応のPCIデバイスが二つ実装された場合に使われるモードです．

〔表1〕従来のPCIとPCI-Xのバスコマンド

C/BE#[3:0]値	PCIバスコマンド名	PCI-Xバス規格・バスコマンド名	データ長	バイトイネーブル値
0000b	Interrupt Acknowledge	Interrupt Acknowledge	32ビット長	アトリビュートフェーズ
0001b	Special Cycle	Special Cycle	32ビット長	アトリビュートフェーズ
0010b	I/O Read	I/O Read	32ビット長	アトリビュートフェーズ
0011b	I/O Write	I/O Write	32ビット長	アトリビュートフェーズ
0100b	(予約)	(予約)	–	–
0101b	(予約)	Device ID Message	バースト	全バイト有効
0110b	Memory Read	Memory Read DWORD	32ビット長	アトリビュートフェーズ
0111b	Memory Write	Memory Write	バースト	データフェーズ
1000b	(予約)	(Alias to Memory Read Block)	バースト	全バイト有効
1001b	(予約)	(Alias to Memory Read Block)	バースト	全バイト有効
1010b	Configuration Read	Configuration Read	32ビット長	アトリビュートフェーズ
1011b	Configuration Write	Configuration Write	32ビット長	アトリビュートフェーズ
1100b	Memory Read Multiple	Split Completion	バースト	全バイト有効
1101b	Dual Address Cycle	Dual Address Cycle	–	–
1110b	Memory Read Line	Memory Read Block	バースト	全バイト有効
1111b	Memory Write and Invalidate	Memory Write Block	バースト	全バイト有効

※ 部分は将来予約済みであり，現在はサポートされていないバスコマンド

●メモリライト
●I/Oリード
●I/Oライト
●コンフィグレーションリード
●コンフィグレーションライト
●割り込み応答
●スペシャルサイクル
●デュアルアドレスサイクル

PCIと同様とはいえ，アドレスフェーズの次にはアトリビュートフェーズが追加されています．また従来のPCIではデータフェーズのC/BE#信号でバイトイネーブルを示していましたが，PCI-XではアトリビュートフェーズのC/BE#信号がバイトイネーブルを示します．

メモリライトはメモリ空間アクセス用，I/Oリード/ライトはI/O空間アクセス用，コンフィグレーションリード/ライトはコンフィグレーション空間アクセス用のコマンドです．また，割り込み応答はx86系チップセット間の割り込みベクタ情報のやり取りに，スペシャルサイクルはバス上に接続されたすべてのデバイスに対しての同報通信用コマンドです．

これらのコマンドは，メモリライトコマンドを除き，32ビット(DWORD)長，つまり1ワードのデータ転送が行われます．そのためシングル転送系コマンドに分類されます．64ビットバスのシステムであっても，有効データ長は32ビットです．

メモリライトコマンドは，従来のPCI同様，PCI-Xでもこのコマンドでメモリライトのバースト転送を行うことができます．よってバースト転送系コマンドに分類します．転送バイト数はアトリビュートフェーズのAD信号とC/BE#信号に格納され，バイトイネーブルはデータフェーズのC/BE#信号で示されます．

なお，最後のデュアルアドレスサイクルは64ビットアドレスを指定する場合に使用するコマンドで，この次にメモリアクセス系のコマンドが必ず発行されます．これ単体で使うことはありえません．

● 名前が変更されたコマンド

従来のPCIと同様の動作ですが，コマンド名称が変更されたものには次のようなものがあります．

●シングルメモリリード(Memory Read DWORD)

従来のPCIではメモリリードコマンドでバースト転送も発生しましたが，PCI-Xではこのコマンドはシングル転送専用に定義されました．よってバイトイネーブルもアトリビュートフェーズのC/BE#信号で示されます．

● PCI-Xで規定されたコマンド

PCI-Xで新たに規定されたコマンドには次のようなものがあります．

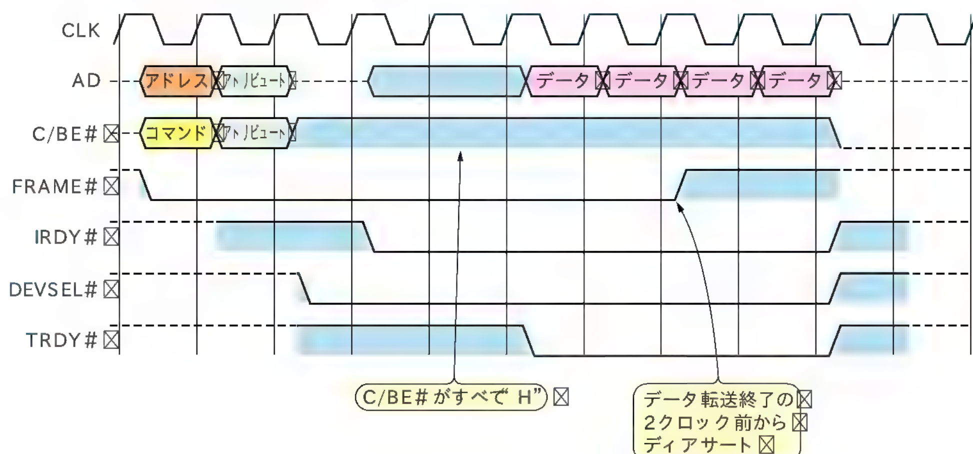
●メモリリードブロック
●メモリライトブロック
●スプリットコンプリーション
●デバイスIDメッセージ

このほか，将来の拡張用に使用するメモリリードブロック(Alias to Memory Read Block)とメモリライトブロック(Alias to Memory Write Block)がありますが，現在は使用されていません．

メモリリード/ライトブロックはその名のとおりメモリ空間のブロック転送で使います．スプリットコンプリーションについては後述します．デバイスIDメッセージはバスブリッジ間でのアクセスの際に使用されるため，ここでは詳細の解説は割愛します．

これらのコマンドはすべてメモリ空間へのトランザクション用コマンドで，しかもバースト転送を行う(転送長が1ワード

〔図5〕メモリリードブロック時(バースト転送)のバスの動作



という場合もある)ため，バースト転送系コマンドに分類されます．これらブロックライトコマンドでは，32ビットまたは64ビットのデータバウンダリすべてがバイトイネーブル有効となります．このため，データフェーズ中のC/BE#信号は用いられません．同じバースト転送系コマンドでも，メモリライトコマンドとはこの点が異なります．

転送バイト数はメモリライトコマンドと同様，アトリビュートフェーズのAD信号とC/BE#信号に格納されています．

5 PCI-Xのバスの動作

● バースト転送の場合

PCI-Xのバストランザクションを，もう少し詳しくみてみましょう．

図5にメモリリードブロックの場合のバスの動作を示します．バースト転送はメモリ空間に対するアクセスでのみ使われます．またアトリビュートとしては，バーストトランザクションアトリビュートが使用されます．

PCI-Xなのでアトリビュートフェーズが増えているのは当然として，従来のPCIと比較して一目で気がつく部分としては，データフェーズのC/BE#がすべて"H"レベルである点，そしてバースト転送終了を示すFRAME#のディアサートが2クロック前に行われている点です．バースト転送はすべてのバイト位置が有効と規定されているので，データフェーズでのC/BE#を参照する必要がなくなりました．このとき，PCI-Xでは"H"レベルを保持するよう規定されています．

バースト転送の中でも一つだけ例外があるのが，メモリライトコマンドです．じつは図3(b)に示したPCI-Xでのデータ転送例は，メモリライトコマンドによるバースト転送時のものです．メモリライトコマンドは，データフェーズでのC/BE#をバイトイネーブルとして参照する必要があります．

● シングル転送の場合

図6に，I/Oライトの場合のバスの動作を示します．バースト転送以外のバスコマンドは，すべてDWORD(32ビット)長のデータ転送，すなわちシングル転送が行われます．アトリビュートフェーズで出力されるアトリビュートは図4(a)になります．

従来のPCIの波形を見慣れていると，まずFRAME#がアサートされ続けているのを見て，バースト転送だと勘違いしそうです．そしてFRAME#やIRDY#より先にDEVSEL#やTRDY#がディアサートされている部分に違和感を感じるでしょう．さらにはFRAME#とIRDY#が同時にディアサートされている部分は，従来のPCIではプロトコル違反とされている動作です．

また，こちらもデータフェーズでのC/BE#はすべて"H"レベルになっています．シングル転送のバイトイネーブルはアトリビュートフェーズのC/BE#で示されます．

● 従来のPCIとPCI-Xの信号線制御方法の違い

従来のPCIと比較し，PCI-Xではトランザクションにアトリビュートフェーズが追加されたこと以外にも，信号線の制御方法が異なる部分があります．まとめると次のようになります．

▶FRAME#の動作が異なる

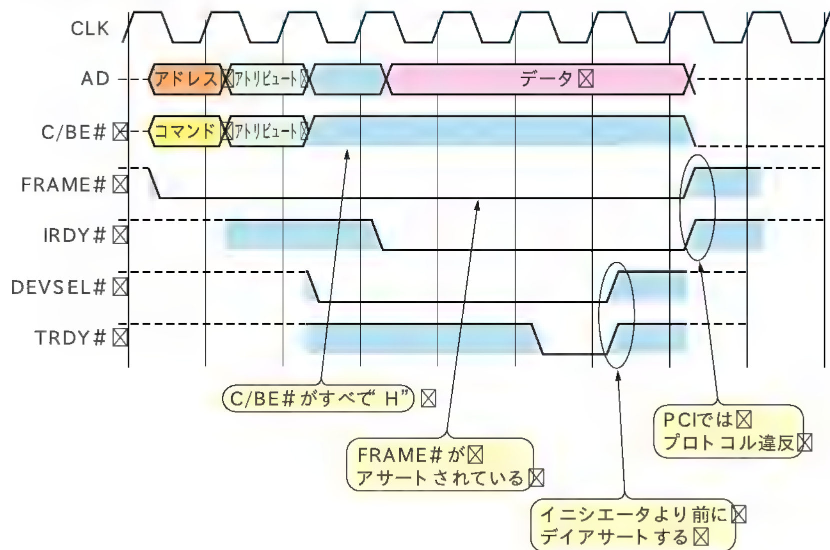
コンベンショナルPCIのシングル転送では，IRDY#がアサートされると同時にFRAME#はディアサートされました．またバースト転送では，FRAME#はアサートしたままで，最終転送ワードの直前にディアサートしました．さらにFRAME#とIRDY#を同時にディアサートするのはプロトコル違反でした．

PCI-Xではシングル転送でもトランザクションの最後までFRAME#をアサートし，FRAME#とIRDY#を同時にディアサートします．またバースト転送では最終転送ワードの2クロック前にFRAME#をディアサートします．

▶TRDY#のアサートタイミングルール

コンベンショナルPCIでは，DEVSEL応答後のTRDY#ア

〔図6〕I/Oライト時(シングル転送)のバスの動作



サートのタイミングは，ターゲット側の都合だけで自由に決定できました．

PCI-XではDEVSEL応答した次のクロックでTRDY#をアサートするか(ノーウェイト)，DEVSEL応答した次のクロックから数えて2クロック後(2クロックウェイト)，または4クロック後(4クロックウェイト)，6クロック後，…というように2の倍数のウェイトを入れたタイミングでしかアサートしてはいけません．これは，ライトサイクル時のイニシエータの出力データのパイプラインを乱さないようにするためです．

▶ バースト転送を開始したらIRDY#やTRDY#をディアサートしてウェイト状態にすることはできない

コンベンショナルPCIでは，イニシエータはIRDY#を，ターゲットはTRDY#を使って，それぞれの都合でウェイト状態を示すためにディアサートすることができました．

PCI-Xではいったんバースト転送を始めたら，途中でウェイトをかけることができません．したがって，イニシエータ側は途中でウェイトを入れなくても転送を続けることができる状態(FIFOの準備など)になってからトランザクションを開始する必要があります．またターゲット側はトランザクションが発行された時点で，そのアクセスの応答準備に多少時間がかかる場合はDEVSEL応答後にターゲットウェイト状態(TRDY#アサートを遅らせる)に入るか，さらに時間がかかるようなら，リトライ，もしくはスプリットレスポンス(詳細後述)を返す必要があります．またバースト転送開始後に何らかの都合でトランザクションを打ち切りたい場合は，後述するルールにしたがって特定のタイミングでディスコネクトを返してトランザクションを打ち切ります．

▶ データフェーズでC/BE#をバイトイネーブルとして参照するのはメモリライトコマンドのみ

メモリライトコマンド以外のバースト転送系コマンドは，すべてのバイト位置を有効と考えるので，バイトイネーブルは不要です．シングル転送のバイトイネーブルはアトリビュートフェーズで出力されるC/BE#をバイトイネーブルとします．よってデータフェーズのC/BE#を参照するのは，メモリライトコマンドのみとなります．

● バースト転送重視のプロトコルへ

PCI-Xはコンセプトとして，いかに効率よくバースト転送し続けるか？ということを主眼においています．従来のPCIで使われた，データフェーズ途中でウェイトが入ったり，FRAME#やほかの信号をみながらバースト転送の終了を認識するような方法は，PCI-Xでは採用されていません．いったんデータ転送が始まったらいくつかの終了条件が成立するまで，あとはノーウェイトでデータ転送を行うという方法に変わりました．

一見すると乱暴なやり方にみえるのですが，これによりイニシエータとターゲットの実際の設計はかなり楽になったということもいえるのです．筆者としては，非常に洗練されたバス規格になったと感じます．

6 トランザクションターミネーション

PCI-Xのトランザクションの終了(ターミネーション)は，従来のPCIに比べていくつか追加されています．表2にターミネーション方法の一覧を示します．ここでは図5や図6では説明しきれない，トランザクションの終了時のルールについて説明します．

● イニシエータターミネーション

イニシエータ側の都合によるトランザクションの終了処理で，次のようなものがあります．

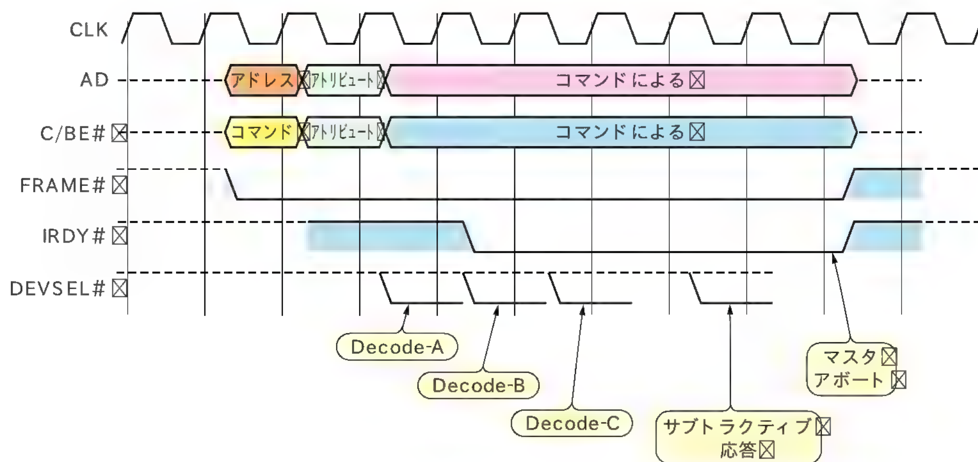
▶ バイトカウントサティスファクション

イニシエータデバイスの都合によるトランザクション終了処

〔表2〕ターミネーション一覧

ターミネーション	DEVSEL#	TRDY#	STOP#	データ転送	トランザクション
マスタアボート	ディアサート	ディアサート	ディアサート	なし	終了
ウェイトステート	アサート	ディアサート	ディアサート	なし	続行
データ転送	アサート	アサート	ディアサート	成立	バースト転送時なら続行，シングル転送時なら終了
ターゲットアボート	ディアサート	ディアサート	アサート	なし	終了
リトライ	アサート	ディアサート	アサート	なし	終了→再発行
シングルデータフェーズディスコネクト	ディアサート	アサート	アサート	成立	終了→再発行
NextADBディスコネクト	アサート	アサート	アサート	成立	終了→再発行
スプリットレスポンス	ディアサート	アサート	ディアサート	なし 注	終了→スプリットコンプリーションコマンド発行

注：ライトサイクル時はターゲットの仕様による



〔図7〕
DEVSEL応答タイミングとマスタアボート

理です．一般的にイニシエータの要求するデータ長のデータ転送が正常に完了した場合には，このターミネーション方式によりトランザクションが完了します．とくにバースト転送時において，規定のデータ転送長の転送が終わった場合には，この完了処理をバイトカウントサティスファクションという名称で呼びます．

▶ マスタアボート

いくつかのターミネーション規定はコンベンショナルPCIから使用しているものがそのまま継承されていますが，このターミネーションもその一つです．

マスタアボートは，イニシエータが発行したトランザクションに対して，いずれのデバイスからもDEVSEL応答が返ってこなかった場合に，イニシエータがトランザクションを打ち切るターミネーションです．

アクセス先のデバイスが応答しないということは，誤ったアドレス空間へのバスアクセスが行われたことを示し，例外処理/異常処理として扱うべき内容のものです．プロセッサでたとえるとバスエラーと同一のものです．

ただし，コンフィグレーションサイクルでは，そのバス番号/デバイス番号/ファンクション番号にデバイスが存在するかどうかを調べる唯一の手段であるため，マスタアボートが必ずしもアクセス異常というわけではありません．

PCI-Xの場合，DEVSEL応答タイミングとしてDecode-A，Decode-B，そしてDecode-Cが規定されています．さらにDecode-Cの2クロック後にサブトラクティブ応答があります（図7）．

FRAME#信号のアサートからDEVSEL応答の待ち時間は，アドレスフェーズ後6クロック以内となり，従来に比べて長く感じられますが，これはターゲット内部でFRAME#の入力やDEVSEL#の出力がクロック同期で行われることによる遅延を考慮したものです．

● ターゲットターミネーション

ターゲット側の都合によるトランザクションの終了処理です．ターゲットターミネーションはデータフェーズ中にのみ発行されるターミネーションであり，ウェイト状態も含めて表2のように，DEVSEL#/TRDY#/STOP#の組み合わせでイニシエータに通知します．

▶ シングルデータフェーズディスコネクト

1ワードのデータ転送が完了した場合において，それ以上のデータ取り込みを行わないということを通知するためのディスコネクトです．

このディスコネクトは，一般的にメモリリード/ライトブロックやメモリライトなどのバースト転送系コマンドが発行された際に，バースト転送へ応答ができないことをイニシエータに通

知するために発行します．これは従来の PCI バスにおけるディスコネクトと同一のものですが，各信号線の制御のしかたは似て非なるものなので注意が必要です．

従来の PCI でのディスコネクトは，FRAME# がディアサートされるまで STOP# をアサートし続けました．また DEVSEL# は STOP# をディアサートするときに同時にディアサートしました．しかし PCI-X のシングルデータフェーズディスコネクトでは，STOP# のアサートと同時に DEVSEL# をディアサートしてしまいます．また STOP# をアサートした次のクロックにはディアサートしてしまいます（図 8）．またシングルデータフェーズディスコネクトはデータ転送を伴うディスコネクトなので，図 8 のように STOP# アサートと同時に TRDY# もアサートします．混乱するのは DEVSEL# をディアサートしているにもかかわらず，これでデータ転送が成立している点です．

シングルデータフェーズディスコネクトは，バースト転送を受け付けない場合だけでなく，バースト転送開始時においてアドレスの並びがターゲットデバイス側には不都合な場合にも活用できます．たとえば，64ビット /8ワードバーストを基本としたメモリを実装している PCI-X デバイスの場合，バースト転送開始アドレスとしては，00h，40h，80h……というような 64バイト単位のアドレスがちょうどよいアドレスになります．ここでバースト開始アドレスとして 18h や 64h のようなアドレスの並び＝アラインが乱れているような場合には，そのアドレスからバースト転送を受け付けることができません．

そこで，アドレスの並びが都合のよい番地になるまではシングルデータフェーズディスコネクトを発行してイニシエータ側にトランザクションの再試行を行わせ，アドレスの並びが都合よくなった場合に一気にデータを取り込む…という動作を行うわけです．

▶ NextADB ディスコネクト

アローワブルデータバウンダリ（ADB）という，128バイト単位のデータを転送した境界で，ターゲットがいったんデータ転送を打ち切るためのディスコネクトです．

PCI-X の場合，バースト転送コマンドが発行された場合のデータの転送方法としては，

1）イニシエータの要求にしたがい，すべてのデータ転送を受け付ける
2）シングルデータフェーズディスコネクトで 1ワード目のみ転送する
3）NextADB ディスコネクトである程度まとまったブロックサイズ単位で転送する

のいずれかの方法を採ります．これ以外で，バースト転送の途中でウェイトをかけたり，ディスコネクトなどでターミネーションする方法はありません．

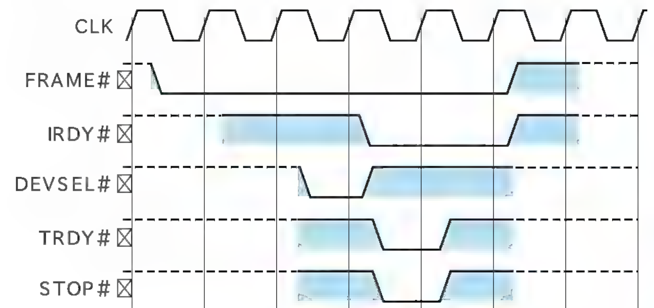
したがって，バースト転送が行われた際に，ターゲット側の都合で転送中にウェイトをかけたい，またはトランザクションを打ち切りたい場合は，3）の方法で 128バイト（32ビット PCI-X バスシステムでは 32ワード，64ビット PCI-X バスシステムでは 16ワード）単位でディスコネクトを行います（図 9）．

NextADB ディスコネクトを利用することで，ディスコネクト後にターゲット内部で次のトランザクションに応答するためデータ受け入れの準備を行う時間を確保できます．

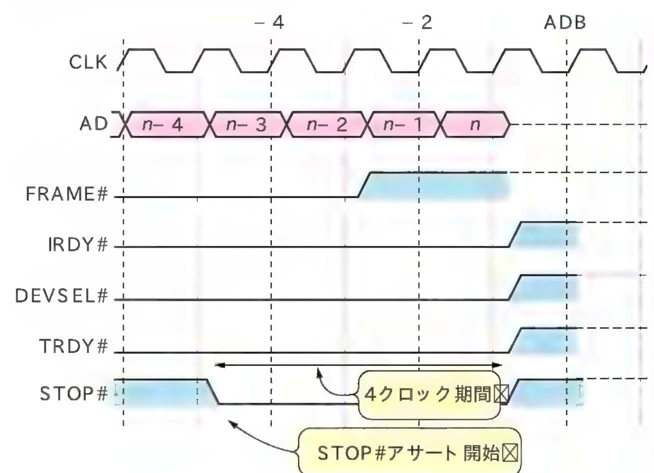
▶ リトライ

現在のトランザクション要求に対して直ちにターゲットが応答できないということで，トランザクションの再発行をイニシエータにお願いしてトランザクションを打ち切るというターミネーションです（図 10）．

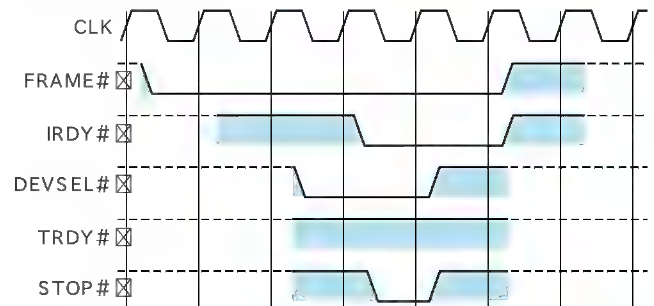
〔図 8〕シングルデータフェーズディスコネクト



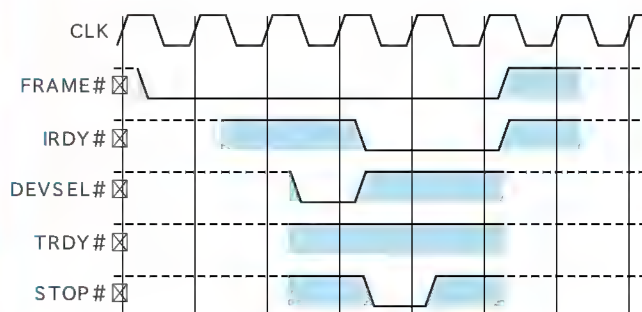
〔図 9〕NextADB ディスコネクト



〔図 10〕リトライ



〔図11〕ターゲットアボート



▶ ターゲットアボート

現在のトランザクション要求が，ターゲットデバイスとしてはまったく受け付けることができないような致命的な場合に応答するターミネーションです（図11）．

たとえば，デバイスの初期化が行われていないにもかかわらずに，ある保護領域に対してアクセスが行われたりなど，例外的なアクセスに際し，ターゲットアボートを返すことでイニシエータに対して異常なアクセスであるということを通知することが可能です．

▶ スプリットレスポンス

現在のトランザクションに対してただちに応答することはできないが，後述のスプリットトランザクションを活用することで効率よくトランザクションを継続できるような場合に使用されるターミネーションです（図12）．

「ただちに応答できないため，再度トランザクションを発生させる」という点はリトライターミネーションと同様の考えなのですが，リトライは再度イニシエータがトランザクションを発行し，ターゲットがそれに答えるという「受動的」な対応なのに対して，スプリットレスポンスターミネーションの場合は，再度発行するトランザクションは自分がコマンドを発行する「能動的」な対応を行います．詳細は次の項目で解説します．

7 スプリットトランザクション

● バス使用効率向上のために

スプリットトランザクションはPCI-Xで追加された新しいバストランザクションであり，高度なPCI-Xアドインカードの設計を検討する場合にはサポートすべき内容のものです．

スプリットトランザクションは，非常にユニークな動きをおこなうトランザクションであり，コンベンショナルPCIにおいてパフォーマンスを向上させる際にネックとなっていた，リードアクセス時のターゲット側のウェイト時間を削減することを目的としたトランザクションです．

● 従来のPCIでは

従来のPCIにおけるデータの読み出しサイクルでは，ターゲット側が読み出しアドレスをバックエンドの回路に渡し，読み出しデータを取り込んでからPCIバス上に出力するというデータの転送経路が存在します〔図13（a）〕．このため，一般的には書き込みサイクルより読み出しサイクルのほうがバスサイクルが長くなる傾向があります．その結果，エージェント間のデータ転送のために他のバスマスタがバスを使用できなくなるということもあり，バスの使用効率が低下する場合がありました．

● リトライする方法もあるが……

これを回避するため，直ちに読み出しデータを用意できない場合には，ウェイト状態のままバスを占有せずに，ターゲットはリトライを返してトランザクションの再発行を要求して，いったんバスを開放する方法もあります．

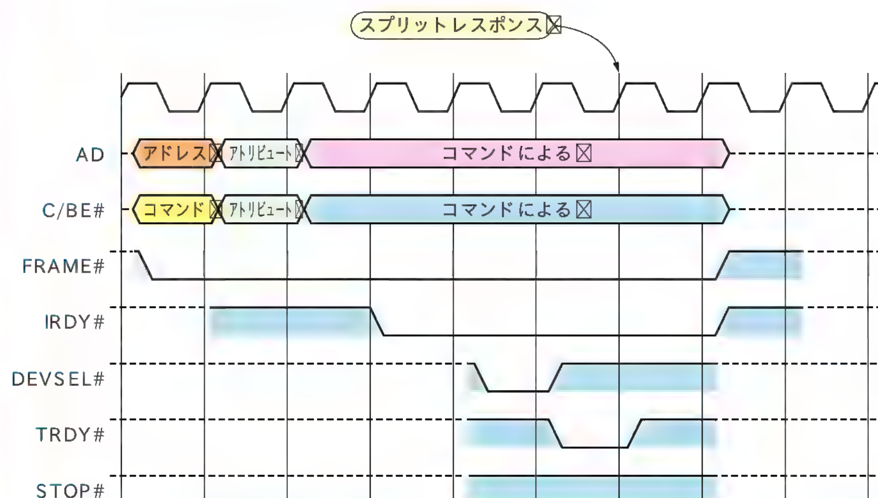
しかしトランザクションの再発行は，ターゲットの読み出しデータの準備完了のタイミングとは無関係に，イニシエータの都合で行われます．したがってトランザクションが再発行されても，ターゲットがまだデータの準備が完了していない場合は，またもリトライを返すことになります〔図13（b）〕．

この方法では，たしかにターゲットがデータを用意するまでの間にバスが開放されますが，ターゲットがデータを用意できたタイミングをイニシエータが知ることができないので，トランザクションの再発行が早ければ再度リトライが返ってきてしまい，遅ければデータ転送のパフォーマンスが下がってしまいます．

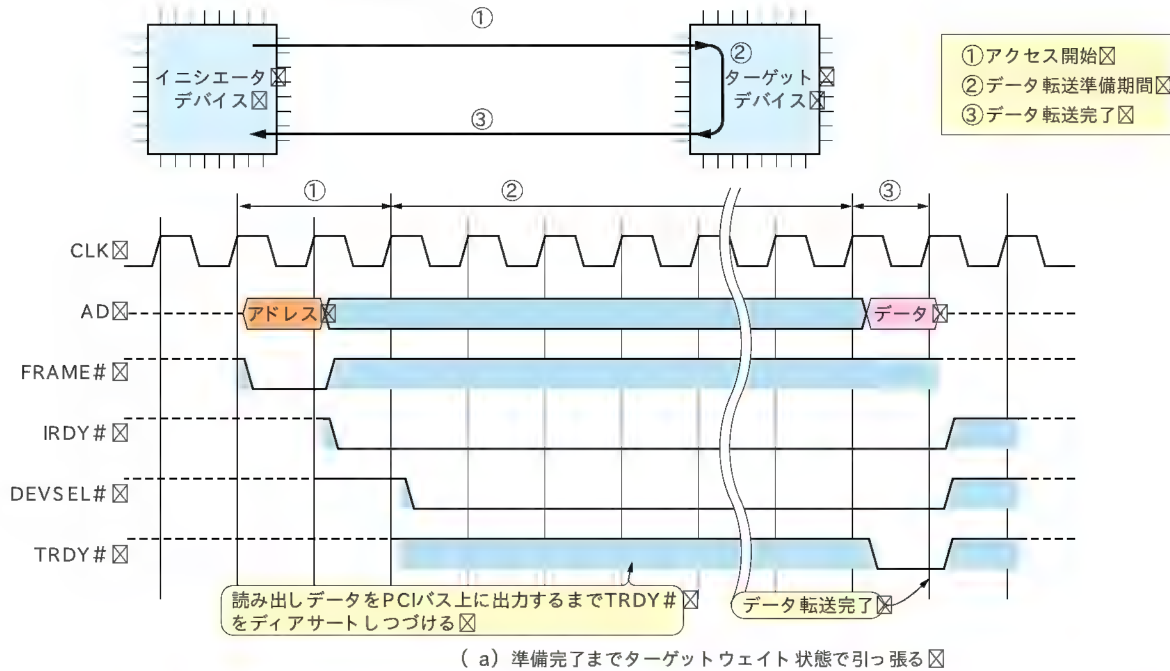
● PCI-Xのスプリットトランザクション

PCI-Xの場合は，コンプリータはリトライの代わりにスプリットレスポンスを返し，トランザクションをいったん打ち切ります．

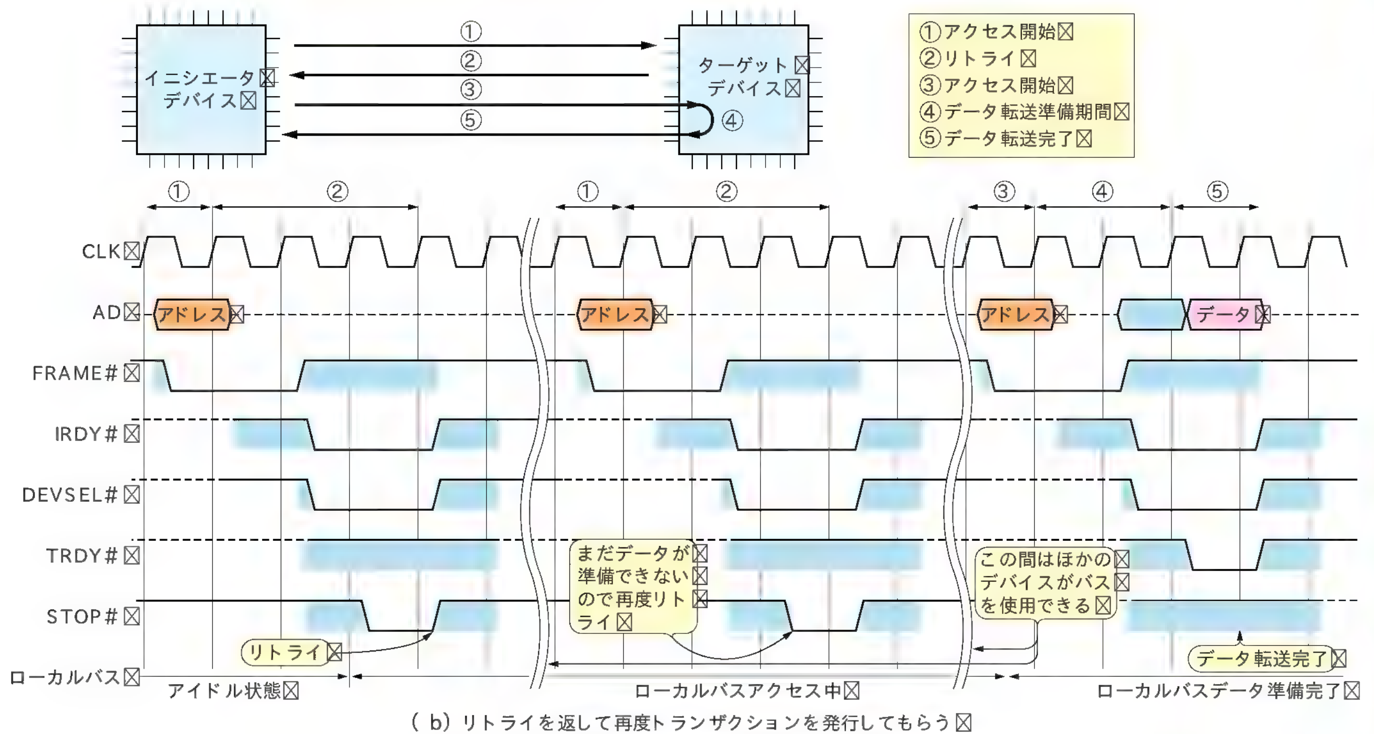
〔図12〕スプリットレスポンス



〔図13〕従来のPCIでのリードサイクルの動き



（a）準備完了までターゲットウェイト状態で引っ張る

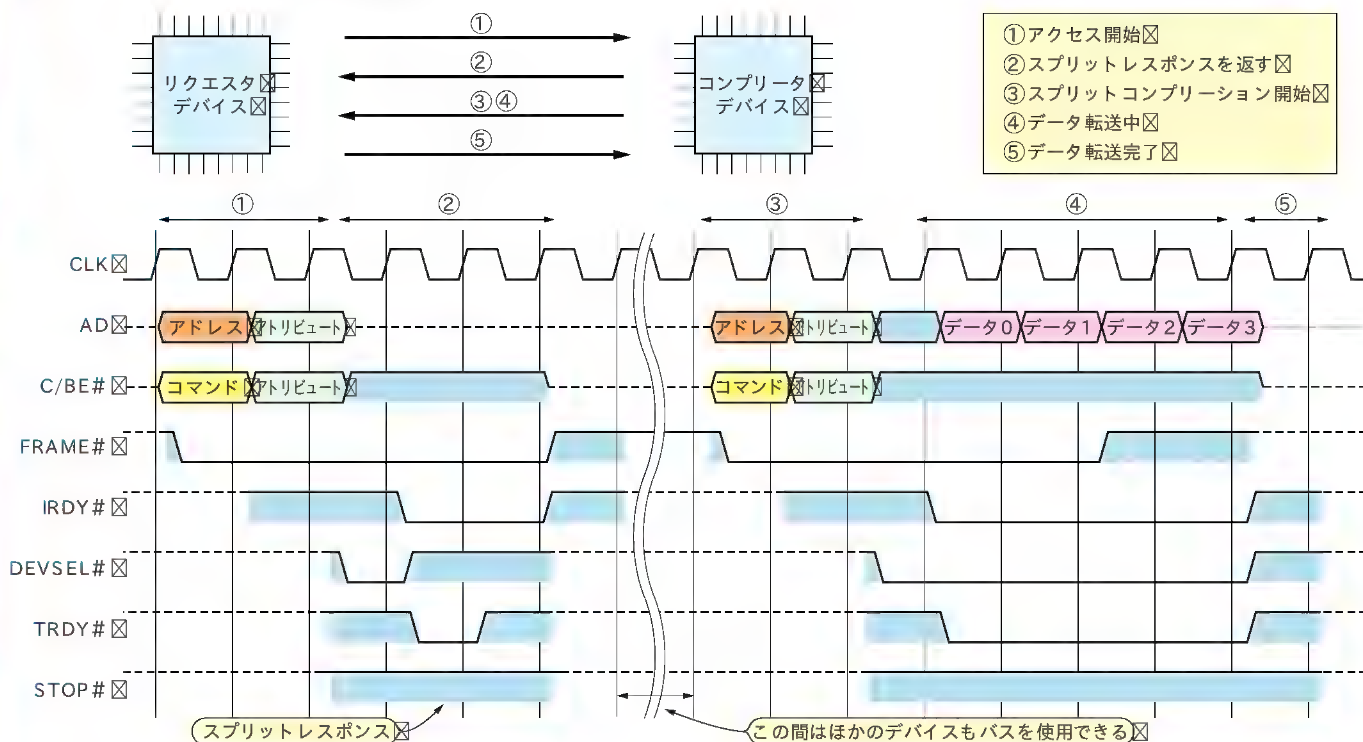


（b）リトライを返して再度トランザクションを発行してもらう

スプリットレスポンスを受け取ったリクエスタは，スプリットコンプリーションコマンドが発行されるのを待ちます．読み出しデータの用意ができたコンプリータは，今度は自らバスの制御権を取得してスプリットコンプリーションコマンドを発行し，さきほどリード要求のあったリクエスタに対して用意できた読み出しデータを書き込みます(図14)．

これにより，データ転送時間が長くなるようなときにも，いったんバスが開放されるため，他のバスマスタがこの間にバスを使用することができます．また，コンプリータがデータ転送の準備ができた時点でトランザクションが開始されるため，トランザクション発行タイミングが早すぎず遅すぎず，最適なタイミングでデータ転送が行われる点が先ほどのリトライを返

〔図14〕スプリットトランザクションの動作



〔図15〕スプリットコンプリーション時のアドレスフェーズとアトリビュートフェース

C/BE[3:0]#	AD[31:00] 31	30	29	28 〜 24	23 〜 16	15 〜 11	10 〜 8	7	6 〜 0
バスコマンド（スプリットコンプリーション）	R	R	RO	タグ	リクエスタ：バス番号	リクエスタ：デバイス番号	リクエスタ：ファンクション番号	R	下位アドレス[06:00]

（a）アドレスフェーズの内容

C/BE[3:0]#	AD[31:00] 31	30	29	28 〜 24	23 〜 16	15 〜 11	10 〜 8	7 〜 0
上位バイト転送数	BCM	SCE	SCM	（未使用）	コンプリータ：バス番号	コンプリータ：デバイス番号	コンプリータ：ファンクション番号	下位バイト転送数

（b）アトリビュートフェーズの内容

注：図中のRは将来拡張用で常時 0' がセットされる

す場合とは大きく異なります．これによりバスの使用効率を向上させることが可能となります

図14でリードコマンドの例を示していますが，スプリットトランザクションは何もリードコマンドだけで使うとは規定されていません．ライトコマンドに対しても使うことができます．つまり，リクエスタからのコマンドが読み出し系のコマンドであれば，スプリットコンプリーションでコンプリータが書き込み動作を行います．逆に書き込み系のコマンドであれば，スプリットコンプリーションでは読み出し動作となります．一般的にはアクセスに時間のかかるリードサイクルで使う場合がほとんどです．

● スプリットコンプリーションの判定

ここで，スプリットレスポンスを返されたリクエスタの動作を考えてください．リクエスタはコンプリータからスプリットコンプリーションコマンドが発行されるのを待つわけですが，単純にバス上にスプリットコンプリーションコマンドが発行されるのを見るわけにはいきません．ほかのエージェントもスプリットトランザクションを行っているかもしれないからです．ではリクエスタはどうやって，自分に対してのスプリットコンプリーションコマンドであるかを判定するのでしょうか．

図15に，スプリットコンプリーションコマンド時のアドレスフェーズとアトリビュートフェーズを示します．アドレス

フェーズを見ると，リクエスタのバス/デバイス/ファンクション番号が格納されていることがわかります．つまりスプリットコンプリーションのアドレスフェーズに出力された値が，自分のバス/デバイス/ファンクション番号と一致した場合に，自分に対してのスプリットコンプリーションであると判定できるのです．

● タグの活用

さらにリクエスタは，スプリットレスポンスに対してスプリットコンプリーションが開始されるのを待たずに，次のトランザクションを開始することもできます．つまり，リード要求だけを次々と発行し，データの準備ができたコンプリータから次々とスプリットコンプリーションでデータを受け取るということも可能になります．

しかしこの場合，場合によっては後から要求したリード要求のデータが先に届くという可能性もあります．リード要求と応答がばらばらになってしまうので，どのリード要求に対する応答データなのかの対応付けが問題になります．

このようなトランザクションの対応付けを判定するために，タグを活用します．あるリード要求にはタグを1に，次のリード要求にはタグを2にします．スプリットコンプリーションによる応答データのアドレスフェーズのタグが1なら，最初に発行したリード要求の応答データであることがわかります．タグを識別すれば，順番が入れ替わっても対応付けを正しく判定できるのです．

● コンプリータの動作

コンプリータ側は，スプリットコンプリーションのアドレスフェーズには，スプリットレスポンスを返したときのトランザクションのアトリビュートの内容をコピーして出力します．これにより，タグも含めて，リクエスタのバス/デバイス/ファンクション番号が出力されるわけです．ただし下位ビット6～0は，下位アドレスを格納してください．

アトリビュートフェーズでは，通常のバスコマンドと同様にバス/デバイス/ファンクション番号を出力しますが，その値はコンプリータ，つまり自分の各番号を入れる点に注意してください．またC/BE#やAD下位ビットには，転送バイト数を格納してください．

スプリットコンプリーションのアトリビュートフェーズがほかのコマンドと異なるのは，ADの上位3ビットです．

ここでもう一度，表1のバスコマンド表を見てください．コンフィグレーションやI/O，メモリコマンドは，ビット0が‘0’のときにリード，‘1’のときにライトになっているのがわかります．しかし，スプリットトランザクションはリードだけでなくライトでも使えると説明しました．しかし，スプリットコンプリーションコマンドは1100bの一つだけです．転送方向はどのようにして指定するのでしょうか．

この転送方向の指定に，アトリビュート中のADビット29のスプリットコンプリーションメッセージ（SCM）ビットを使います．バスコマンドと同様，このビットが‘0’であればリード，‘1’であればライトになります（BCMやSCEについての詳細は省略）．

● スプリットコンプリーションコマンドでの注意点

スプリットコンプリーションコマンドに応答するリクエスタは，そのトランザクションのターミネーションにさらに条件が付けられ，シングルワードディスコネクトやデータ転送中のディスコネクト，リトライを返すことはできません．唯一許されているのは，NextADBディスコネクトでデータ転送をいったん打ち切ることだけです．それ以外では，コンプリータからのすべてのデータ転送を受け付けなければなりません．

もっともスプリットコンプリーションは，リクエスタからの要求にコンプリータが応答するトランザクションですから，自分から「アドレスxxxからyyバイト分データをくれ」といっておきながら，データを用意してきたコンプリータに対してリトライなどを返すのはおかしいわけです．またNextADBディスコネクトについても，はじめから自分がノーウェイトで受け取れるだけのバイト数だけを要求すれば，ディスコネクトしなくてもすむはずです．

● コンプリータ（ターゲット）側にはバスマスタ機能が必要

コンベンショナルPCIでは，自分からバスの制御権を取得しないターゲット機能のみのPCIデバイスもありましたが，PCI-Xでスプリットトランザクションをサポートするためには，イニシエータ/ターゲット両方のエンジンが必要になります．

第3章のPCI-Xデバイス設計の解説では，PCI-Xでもっとも基本的な最低限のトランザクションを理解することを第一としたため，スプリットトランザクション機能は実装していませんが，いずれ機会があるときに，FPGAでスプリットトランザクションをサポートしたより本格的なPCI-X対応デバイスの設計事例を解説したいと考えています．

8 コンフィグレーションサイクル

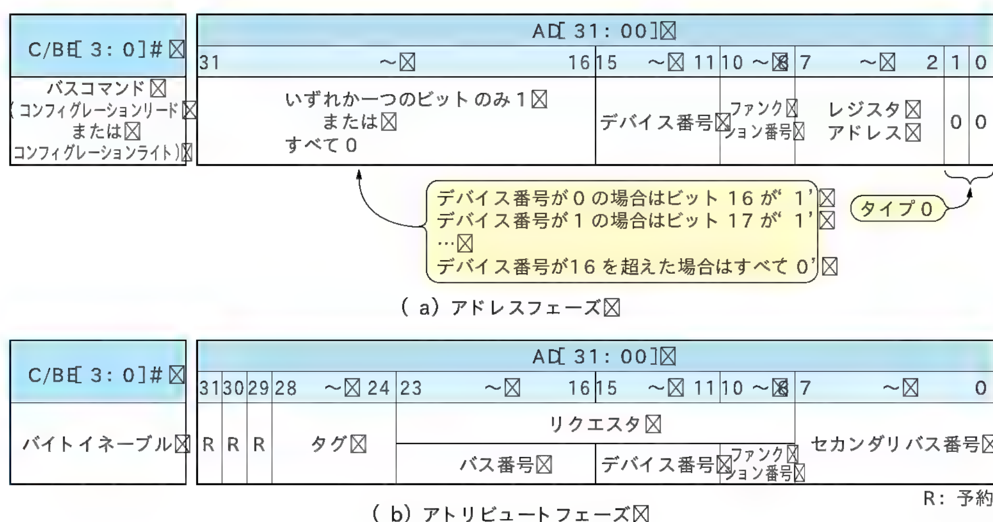
● 自分が実装されているバス番号/デバイス番号を知るには？

各コマンドのアトリビュートフェーズでは，リクエスタのバス番号やデバイス番号などが格納されています．スプリットコンプリーションではコンプリータもバス番号やデバイス番号を出力します．

つまりPCI-Xでは，各デバイスが，自分が何番のバスに接続されているか，デバイス番号は何番が割り当てられているかを知る必要があります（ファンクション番号はそのデバイス設計時に決定されるので調べる必要はない）．

PCI-Xでは，コンフィグレーションコマンドにおけるアドレスフェーズとアトリビュートフェーズに，これらの情報が出力されるように規定されました．

〔図16〕コンフィグレーションサイクル時のアドレスフェーズとアトリビュートフェーズ(PCI-X/Mode1)



ここでは通常のデバイスで必須な，タイプ0のコンフィグレーションサイクルについて説明します．

● アドレスフェーズとアトリビュートフェーズのフォーマット

図16に，コンフィグレーションサイクル時のアドレスフェーズとアトリビュートフェースを示します．

従来のPCIでは，アドレスフェーズではビット10～0までが規定されていましたが，PCI-Xでは図16(a)のように32ビット分すべてが明確に規定されました．これにより，自分に割り当てられたデバイス番号(とファンクション番号)を取得することができます．

さらにアトリビュートフェーズには，ほかのPCI-Xのバスコマンドと同様に，リクエスタのバス番号やデバイス番号が格納されています．タグも同様に存在しますが，ビット31～29はほかのコマンドと異なりすべて予約と規定されています．そして重要なのが，ビット7～0格納されているセカンダリバス番号です．

コンフィグレーションサイクルでのリクエスタとは，すなわちホスト-PCIブリッジ，またはPCI-PCIブリッジの下に接続されている場合はPCI-PCIブリッジを示します．ホストブリッジやPCI-PCIブリッジに対してセカンダリバスといえば，すなわちコンプリータが接続されたバス番号を意味するわけです．

よって，コンフィグレーションサイクルのアドレスフェーズでデバイス番号(とファンクション番号)を，アトリビュートフェーズでバス番号を取得できるわけです．内部ではこれを保持しておき，トランザクションを発行する場合にアトリビュートフェーズに出力する値を生成します．

まとめ

PCI-Xバス規格は，従来のPCIバス規格の信号を使いながら効率よくデータ転送を行うことを目的として設計されたバス規格です．その思想は，単にバースト転送を多用したデータ転送方式や，クロック周波数を上げて転送帯域を向上させるというものだけではなく，デバイスの設計のしやすさにも考慮している点が見受けられます．

たとえば，

- トランザクション開始時点でバースト転送長が確定
- バスコマンドによってシングル/バーストの区別ができる
- いつバースト転送が終わるのかを明確にできる

ということで，ステートマシンを作りやすくしています．

またデバイスの物理的な点については，各信号をいったんクロックで同期化して入出力することで，デバイス内部でFRAME#やIRDY#などの制御信号のセットアップ/ホールドタイムを稼ぐことができます．

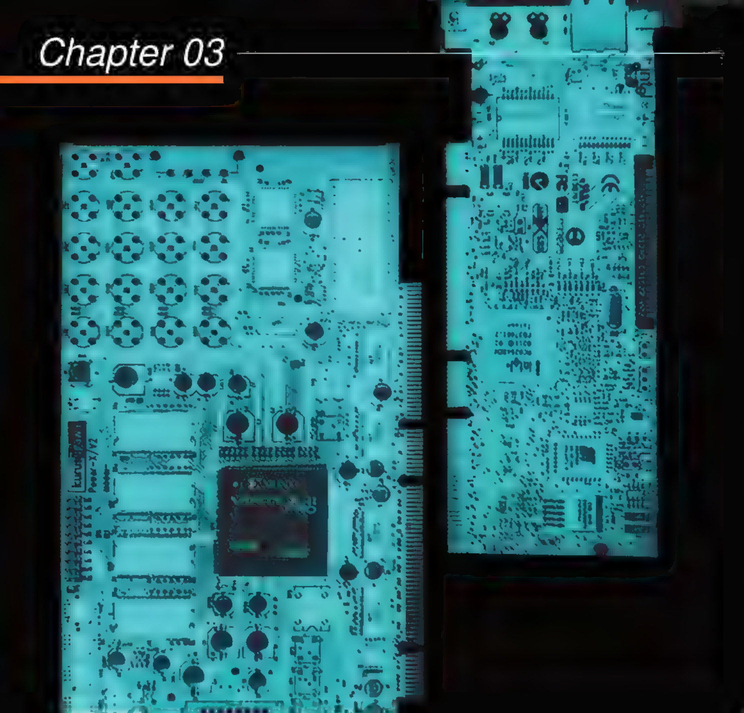
現在のところPCI-X対応のマザーボードは，企業などのサーバ用途だけでなく，個人所有のサーバ向けにも着実に広がっており，自作マシンを販売している店頭でも比較的容易に購入できるようになりました．

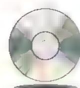
本特集記事が，PCI-Xバスの普及に一役買えれば幸いです．

参考文献

1) PCI-X Protocol Addendum to the PCI Local Bus Specification Revesion 2.0a，PSI-SIG

いくら・まさみ　来栖川電工有限会社





64ビット PCI-X評価ボードおよび
32ビット PCI評価ボードを使って実現する

FPGAによるPCI-X対応デバイス設計事例

井倉 将実

この章では，いよいよFPGAによるPCI-X対応デバイスの設計事例について解説する．PCI-Xアドインカードは64ビットデータバスで，133MHzというクロックに対応するために高性能なデバイスが必要と思われるが，じつは32ビットデータバスのPCI評価ボードを改造することで，手軽にPCI-Xの評価を行うことも可能である．ここでは64ビットデータバスのPCI-X評価ボードと，32ビットデータバスのPCI評価ボードを改造したボードの2種類を使い，実際に動作するPCI-Xターゲットデバイスを設計してみる．
（編集部）

はじめに

今回PCI-X対応デバイスの設計事例を解説するにあたり，筆者の会社では，この特集記事にあわせて「Power-X/V2P」というPCI-X対応評価ボードを開発しました．しかしコラム1でもわかるように，仕様的に高機能なため，手軽にPCI-Xを勉強してみたいという用途には少し荷が重いところがあります．

この特集についての本誌編集者との打ち合わせでも，もっと手軽にPCI-Xを評価できないものかと知恵を絞りあいました．そんなときの，

「32ビット幅のPCI-Xって，ありですかね？」

という何気ない一言がきっかけでした．Stratix評価キット（CQ出版）を改造して，PCI-X対応にしてしまうおうというアイデアを，半信半疑ながら実行してみたところ，うまくいったというのが正直なところです．

本章では，このPower-X/V2PとStratix評価キットの2種類の製品に，実際に動作するPCI-Xターゲットコアを実装し，PCI-X対応デバイスの設計事例について解説します．

1 PCI-Xの信号について

● PCI-Xの信号

PCI-XはPCIバスと互換性があるので，信号線も原則として従来のPCIバス（仕様書ではコンベンショナルPCIと呼ぶ）と互換性があります．表1（p.65）にPCI-Xのバス信号を示します．表を見てもわかるように，そのほとんどが従来のPCIバスで使用されていたものであることがわかります．しかし新しく追加された信号や，使用方法が変わったものもあります．

PCI-X 1.0の時点で追加された信号として，PCIXCAPがあります．そのアドインカードがPCI-Xに対応しているかどうかを識別するときに使う信号です．

さらにPCI-X 2.0では，Mode2でより信頼性の高いデータ転送を行うために，エラー検出方法として従来までのパリティではなくECCが採用されました（PCI-X/Mode1でもオプション仕様として使える）．そのため従来予約とされていた信号がECC［5：2］に割り当てられました．またECC［0］にはPARが，ECC［7］にはPAR64#が名前を変えて使われます．さらにACK64#をECC［1］として，REQ64#をECC［6］としても使い，ECCは合計8ビットとなりました．REQ64#とACK64#は，64ビットデータ転送を行うかどうかをトランザクション開始時点で判定するために使うので，いったんデータ転送が始まると不要な信号になります．そこでデータ転送中はECCとして使おうというものです．

● PCI/PCI-X対応モードの判定

コンベンショナルPCIの66MHzモードも含め，アドインカードがPCI-Xのどのモードに対応しているかを判定する方法について説明します．

コンベンショナルPCIの66MHzモードを示すM66ENピンはサイドBの49番ピンに割り当てられています．33MHz専用の場合はこのピンはGNDに配線されています．66MHz対応の場合はこのピンをNC状態にします．

PCI-X対応デバイスでも，コンベンショナルPCI上で動作させる場合には33MHz専用にする場合も考えられます（PCI-X対応だからといって，コンベンショナルPCIで66MHz動作をさせなければならないわけではない）．このような場合は，PCI-Xアドインボードといえども，このピンはGNDに接続しておきます．

PCI-Xの対応モードを示すPCIXCAPピンは，サイドBの38番ピンに割り当てられています．コンベンショナルPCI専用の場合は，このピンはGNDに配線されています．またPCI-Xでもクロック66MHzと133MHzの2種類がありますが，66MHzまでに対応している場合はこのピンを10kΩでプルダウンしておきます．133MHzまでに対応している場合はNC状態にしておきます（表2，p.65）．なお，各ピンとGNDの間に0.01μFのコンデンサを実装しておくとよいでしょう．

各種モードにジャンパの設定だけで対応できるアドインカー

Column 1

Power-X/V2P について

Power-X/V2P は，ザイリンクス製の最新鋭 FPGA である Virtex-2/PRO を採用し，64ビットデータバス PCI/PCI-X に対応した PCI-X バスシステム開発支援ボードです（写真 A）.

PCI/PCI-X コアを実装する FPGA には，XC2VP7-6FF896CS または XC2VP20-6FF896CS を搭載しています．これらの FPGA はそれぞれ約 100万ゲート，300万ゲート相当（おおまかなゲート換算）の規模をもちます．

この FPGA の最大の特徴は，PowerPC405コア（IBM）があらかじめ実装されている点です．これにより，外付けの CPU を搭載せずとも，FPGA 内部の PowerPC プロセッサを使用して，大規模なソフトウェアを駆使した機器制御を行うことができます．

さらに Power-X/V2P には，データバス幅 64ビットで 1Gビット（128M バイト）の DDR-SDRAM を搭載しています．この DDR-SDRAM はバスクロック 166MHz の通称 DDR333という規格のものです．差動のクロックラインは FPGA 内の DCM によって生成した DDR メモリの駆動クロック（DDRCLK/DDRCLK＊）を供給しています．

また DDR-SDRAM は，本ボード上に実装した 2.5V 電源生成の DC-DC コンバータと，その 1/2の電圧を生成するコンパレータ内蔵のレギュレータによる 1.25V 電源の 2電源で駆動します．そして 1.25V 電源は，FPGA にも V_{ref}（リファレンス電圧）として，DDR-SDRAM の I/O バンクの VREF ピンにも供給され，DDR-SDRAM に接続するすべての I/O ピンを SSTL2-I インターフェースで使っています．

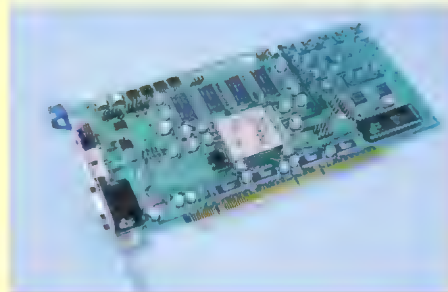
DDR-SDRAM の機能や DDR 対応メモリコントローラを使用する上でのテクニックについては，姉妹誌デザインウェーブマガジン誌の 2003年 11月号で解説しています．

このほか，1チャネルのシリアルポート，フラッシュメモリ，フルカラーアナログ RGB ビデオ出力機能，各種ステータス LED やディップスイッチ系も搭載されており，最終的には PCI-X バスインターフェースをもつ，1ボードコンピュータの実現をめざしています．

なお，今回の特集は PCI-X に関するものなので，ここでは PowerPC405コアや DDR-SDRAM の使用事例は解説しません．機会があれば詳しく解説したいと考えています．

■問い合わせ先

●来栖川電工有限会社
http://www.kurusugawa-ele.co.jp/


〔写真 A〕Power-X/V2P の外観

ドを設計するには，図 1のようにピンを配線するとよいでしょう．またそのジャンパの設定をデバイス側でも識別できるようにするには，PCI カードエッジ側への配線だけでなく，デバイス自身にも配線するのがよいでしょう．

● コンフィグレーションレジスタの処理

コンベンショナル PCI の 33MHz と 66MHz では，コンフィグレーションレジスタのステータスレジスタのビット 5（M66EN）に，66MHz に対応しているかどうかを示すビットが割り当てられています．66MHz に対応する場合は，ステータスレジスタのこのビットも‘1’にセットしておきましょう．

なお過去の経験では，M66EN ピンを NC にしておきながら，ステータスレジスタでは M66EN ビットを‘0’固定にしておいても，66MHz 対応のマザーボードではコンベンショナル PCI の 66MHz で動いていました．供給クロック周波数の決定は，プラットホームの電源投入初時の初期化段階でハードウェア的に判定していて，コンフィグレーションレジスタの内容は確認していないということでしょう．

PCI-X/Mode1では，コンフィグレーションレジスタについてはとくに拡張された機能はありません．

● 動作モードの判定

PCI-X はコンベンショナル PCI バス上でも動作できなければなりません（最低限 33MHz で動作できること）．よって，自分が PCI-X に対応しているからといって，必ずしも PCI-X で動いているとは限りません．

実際のデバイス設計の項目で詳しく解説しますが，コンベンショナル PCI と PCI-X では，各信号線の制御方法を変えなければなりません．つまりデバイスは，どちらのモードで動いているかを知る必要があるのです．

このため PCI-X では，PCI バスのリセット信号の立ち上がりで，TRDY# と STOP#（PCI-X のモードによってはさらに DEVSEL# と PERR# も使う）の信号線の状態の組み合わせで，どのモードで動作しているのかを判定できるようにしています（表 3）.

コンベンショナル PCI でも，66MHz 動作と 33MHz 動作の 2種類があるので，今回の設計では，表 3で示すように M66EN ピンの状態もあわせて確認して動作モードを判定しています．

● クロック周波数判定の必要性

コンベンショナル PCI と PCI-X では信号の制御方法などが異なるので，ここでモード判定が必要なのは理解できます．しかし一般的に，133MHz で動作する PCI-X のデバイスが 66MHz で動かないはずはありません（高速なデバイスを低速で使う）．動作周波数を判定する必要はあるのでしょうか？

〔表1〕PCI-Xバス信号

PCI/PCI-Xバス信号一覧				
信号グループ	信号名	ドライブ方式	本数	用途/備考
クロック	CLK	t/p	1	バスクロック
リセット	RST#	t/p	1	システムリセット
32ビットバス	AD[31:00]	t/s	32	アドレス/下位データバス
	C/BE#[3:0]	t/s	4	コマンド/下位バイトイネーブル
64ビットバス	PCIAD[63:32]	t/s	32	アドレス/上位データバス
	C/BE#[7:4]	t/s	4	コマンド/上位バイトイネーブル
	REQ64#(ECC[6])	t/s	1	64ビットバス転送要求
	ACK64#(ECC[1])	t/s	1	64ビットバス転送応答
バスマスタ機能	REQ#	t/s	2	バスリクエスト
	GNT#	t/s		バスグラント
制御信号	FRAME#	t/s	7	サイクルフレーム
	IDSEL	t/s		初期化デバイスセレクト
	DEVSEL#	t/s		デバイスセレクト
	IRDY#	t/s		イニシエータレディ
	TRDY#	t/s		ターゲットレディ
	STOP#	t/s		ストップ
	LOCK#	t/s		バスロック
割り込み	INTA#	o/d	4	割り込みピンA
	INTB#	o/d		割り込みピンB
	INTC#	o/d		割り込みピンC
	INTD#	o/d		割り込みピンD
信頼性確保エラー報告	PAR/ECC[0]	t/s	1	パリティビット
	ECC[5:2]	t/s	4	PCI-X/Mode2のみ
	PAR64(ECC[7])	t/s	1	パリティ(64ビット用上位32ビット)
	PERR#	t/s	2	パリティエラー通知
	SERR#	o/d		システムエラー検出
特殊機能	PRSNT1#	t/s	2	電源容量通知
	PRSNT2#	t/s		
	M66EN	t/p	1	66MHz対応
	PCIXCAP	t/p	1	PCI-X対応
	PME#	t/p	1	パワーマネージメント

注　t/s：トライステートドライブ
t/p：トーテムポールドライブ
o/d：オープンドレイン

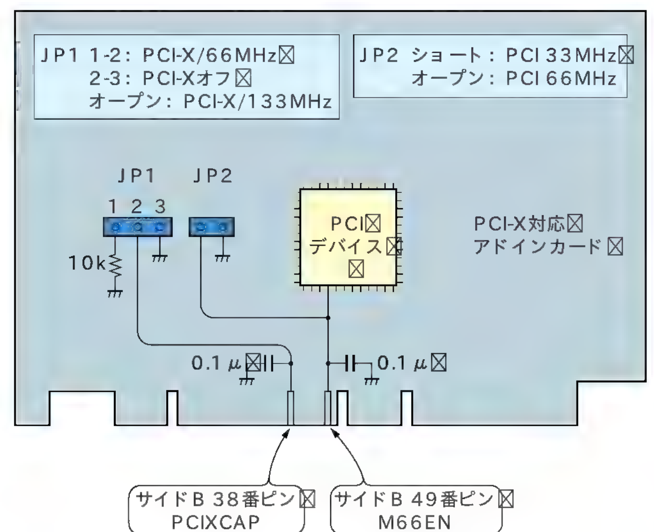
〔表2〕PCI/PCI-X動作モードの設定

PCIXCAP	M66EN	動作モード
GND	GND	コンベンショナルPCI/33MHz
GND	NC	コンベンショナルPCI/66MHz
10kΩプルダウン	未使用	PCI-X/66MHz
NC	未使用	PCI-X/133MHz

〔表3〕動作モードの判定

M66EN	STOP#	TRDY#	モード	最大クロック周期(ns)	最小クロック周期(ns)	最小動作周波数(MHz)	最大動作周波数(MHz)
"L"	"H"	"H"	コンベンショナルPCI/33MHz	∞	30	0	33
"H"	"H"	"H"	コンベンショナルPCI/66MHz	30	15	33	66
–	"H"	"L"	PCI-X/Mode1	20	15	50	66
–	"L"	"H"	PCI-X/Mode1	15	10	66	100
–	"L"	"L"	PCI-X/Mode1	10	7.5	100	133

〔図1〕PCI/PCI-X動作モードのジャンパ設定例



たとえば，SDRAMやDDR-SDRAMの駆動クロックをPCIバスのクロックと共通，または分周や逓倍したものを使用するのであれば，そのクロック周波数が変わるということは，リフレッシュコマンドの発行周期も変わってくることを意味しています．

表4は，今回設計したPower-X/V2Pに搭載したDDR-

SDRAMのリフレッシュ周期を，PCIバスクロックで数えた場合に何クロック分であるかを表したもので，DDR-SDRAMで規定のある約7.8μsのリフレッシュ周期（t_{RP}）から逆算したものです．この表からもわかるとおり，定期的に発行する必要性があるリフレッシュサイクルをサポートするためには，いまバスクロックが何MHzで動作しているのかを識別できなければ，最適なリフレッシュ回路を設計できません．

もちろん，想定される最低クロック周波数（今回の場合，PCI-Xモード時であれば50MHz）を基準として設計し，クロックが上がる場合はリフレッシュサイクルが短くなってもよいとする設計もあるでしょう．しかし，リフレッシュ中はメモリにもアクセスできなくなるわけで，必要以上に短い間隔でリフレッシュを発行してしまうと，それだけメモリアクセスのパフォーマンスが落ちてしまうことになります．

2 Stratix評価キットをPCI-X対応化する

● PCI評価キットをPCI-X上で使うために

今回設計したPower-X/V2Pでは，M66ENピンやPCIXCAPピンをディップスイッチで任意に設定できるように設計しています．つまり，はじめからコンベンショナルPCIの66MHzや，PCI-Xの66MHzや133MHzでの評価を目的に設計しています．ほかにもDDR-SDRAMや，また他社からもFPGAを搭載したPCI-X対応評価ボードが発売されはじめています．しかし一般的に，これらの評価ボードはそれなりに高価です．もっと手軽にPCI-Xを評価できないものでしょうか？

そこで，Stratix評価キットを使ってPCI-Xの評価をしてみたいと思います．しかしStratix評価キットは，出荷時の状態は32ビット/33MHzのコンベンショナルPCIアドインカードであり，そのままではPCIXCAPやM66ENがGNDに直結されているために，PCI-Xモードになりません．

● PCI評価キットをPCI-X上で使うために

そこで，M66ENピンのあるサイドBの49番ピンと，同じくPCIXCAPピンのあるサイドBの38番ピンをカッターナイフや細いドリルなどでパターンカットし，オープンの状態にします（写真1）．

この改造で，PCI-Xには非対応ではあるがコンベンショナルPCIの66MHz対応のマザーボードでは，PCI Rev2.3準拠の32ビット/66MHz対応PCIアドインカードとして使用できます．

同様に，PCI-Xに対応したシステムの上では，PCI-X 2.0準拠のPCI-X/Mode1の32ビット/133MHz対応PCIアドインカードとして使用することができます．

さらに，PCIXCAPピンとGNDの間に10kΩのプルダウン抵抗を接続すれば，PCI-X/Mode1の66MHz対応のアドインカードとして動作させることも可能です．

なお，ボードにこのような改造をすると，当然ながらユーザーサポートは受けられなくなります．その点を考慮し，あくまでユーザー各自の判断で改造を行ってください．

Column 2
メモリリードブロック/メモリライトブロックコマンド

じつは当初，PCI-Xのターゲットコントローラの設計の際には，コンフィグレーションリード/ライトとメモリリード/メモリライトの四つのコマンドしかサポートしていませんでした．初めてのPCI-Xデバイス設計の解説記事という性格上，バースト転送の解説は省き，仮にバースト転送要求が来てもシングルデータフェーズディスコネクトを行ってシングルワードアクセスだけを行い，理解しやすい設計例をと思ったからです．

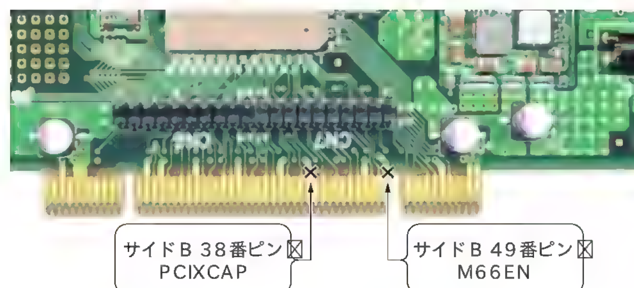
しかしながら，今回Appendix 3で解説があるようにバージョンアップしたPCIデバッグライブラリ for DOSを使って64ビットや128ビット，またはPCIメモリ空間をキャッシャブル空間に設定してメモリの読み書きを行ったところ，PCI-XシステムではCPUのアクセスでもブロックリードコマンドなどが発行されていることが観測されたのです．

メモリリードブロック/ライトブロックコマンドはPCI-Xバスの性能を引き出す本領発揮のバスコマンドですが，通常のCPUがPCIメモリ空間にアクセスしても発行されることはありません．しかし，SIMD系命令やキャッシュ制御機能を活用することで，CPUアクセスの場合でもPCI-Xバス上にメモリブロック系のコマンドが発行されるようです．

〔表4〕DDR-SDRAMのリフレッシュサイクル発行までのカウント値

駆動クロック（MHz）	インターバルタイマ値	DDRリフレッシュ周期（nS）
33	258	7818.18
66	515	7803.03
100	780	7800.00
133	1038	7804.51

〔写真1〕Stratix評価ボードをPCI-X対応化する改造



〔表5〕ターゲットデバイス基本仕様

▶ 実装先基板
●Stratix 評価キット アルテラ社FPGA 搭載 Stratix EP1S10-7FF780 ●来栖川電工社 Power-X/V2P-Model7 ザイリンクス社FPGA 搭載 Virtex2/PRO XC2VP7-6FF896
▶ 搭載機能
●PCI-X ターゲットコントローラ シングル転送のみに対応．バースト転送はシングルデータフェーズディスコネクトによりバスサイクルの打ち切りを通知 ●コンフィグレーションレジスタ搭載 ●FPGA 内部の内蔵メモリをアクセス対象とする

〔表6〕タイプ PCI-X バスコマンド

C/BE#[3:0]値	PCI-X バスコマンド名	データ長	バイトイネーブル値
0000b	Interrupt Acknowledge	32ビット長	アトリビュートフェーズ
0001b	Special Cycle	32ビット長	アトリビュートフェーズ
0010b	I/O Read	32ビット長	アトリビュートフェーズ
0011b	I/O Write	32ビット長	アトリビュートフェーズ
0100b	(reserved)	―	―
0101b	Device ID Message	バースト	全バイト有効
0110b	Memory Read DWORD	32ビット長	アトリビュートフェーズ
0111b	Memory Write	バースト	データフェーズ
1000b	(Alias to Memory Read Block)	バースト	全バイト有効
1001b	(Alias to Memory Read Write)	バースト	全バイト有効
1010b	Configuration Read	32ビット長	アトリビュートフェーズ
1011b	Configuration Write	32ビット長	アトリビュートフェーズ
1100b	Split Completion	バースト	全バイト有効
1101b	Dual Address Cycle	―	―
1110b	Memory Read BLOCK	バースト	全バイト有効
1111b	Memory Write BLOCK	バースト	全バイト有効

3 PCI-X ターゲットシステムの設計

PCI-X 対応のアドインカードを設計する上で非常に楽な点は，PCIXCAP ピンを除くほとんどの信号がコンベンショナル PCI と同一であるということです．

これは何を意味しているかというと，PCI-X に対応した製品を設計したいと思った際に，アドインカード上に搭載されているデバイスが FPGA のようなプログラマブルデバイスの場合には，PCIXCAP（と必要であれば M66EN）ピンを適切に処理するだけで，PCI-X 対応のアドインカードに変更することができるということです（もちろんそのデバイスも PCI-X に対応できるだけの電気的特性やデバイス速度であることも必要）．

ここでは実際に PCI-X のターゲットコントローラを設計し，Power-X/V2P ボートと Stratix 評価ボードにそれぞれインプリメントして，実際に波形観測をするところまで解説していきます．

● ターゲットデバイスの基本仕様

はじめて PCI-X 対応デバイスを設計するという場合を考え，今回はもっとも仕様の簡単な，いわゆるターゲット機能のみ（しかもシングル転送専用）の PCI-X デバイスを設計してみます．

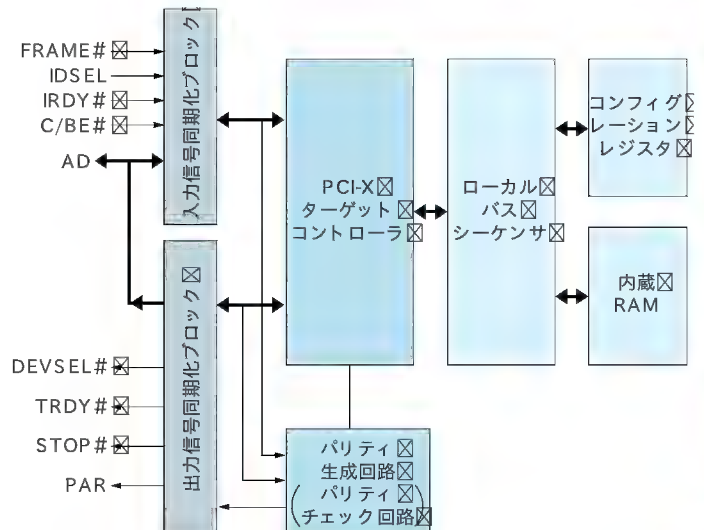
PCI-X の採用を考えるような場合は，性能を追求した場合が多いはず．であるならバスマスタ転送ができなければ PCI-X の意味がない…，せめてバースト転送には対応しないと…，という声はごもっともです．しかし，まずは PCI-X のプロトコルの基本をおさえることが大事です．

参考文献 3）でも，最初に設計する PCI デバイスはコンフィグレーションレジスタのない書き込み専用デバイスでした．PCI-X の場合にはここまで機能を削ることはできませんが，スプリットトランザクションやバースト転送，バスマスタ転送機能などを取り去った，もっともシンプルな PCI-X デバイスを設計することで，PCI-X の基本が理解しやすくなるはずです．

今回設計した PCI-X ターゲットコントローラの基本仕様を表5に示します．また，今回の PCI-X ターゲットコントローラがサポートしているバスコマンド一覧をあわせて表6に示します．

今回バックエンドに用意するのは，FPGA の内部メモリ機能を使った 16K バイトの RAM です．今回想定している Stratix や Virtex-2/PRO の内蔵 RAM は同期式メモリとなります．

以上をまとめたブロック図を図2に示します．



〔図2〕ターゲットデバイスブロック図

● バスモード/クロックの判定方法

PCI-X アドインカードは，コンベンショナル PCI 上でも動作できなければなりません．リセット解除の時点で，バスがどのモードで動作しているかを判定する必要があります．

リスト1が各種モードの判定部分の回路です．リセット信号をクロックで同期化して立ち上がりエッジを検出し，そのとき

〔リスト 1〕バスモード/クロックの判定回路

```
architecture ModeDetect of PCITGT_mdl is

    -- PCI/PCI-Xモード＆クロックレジスタ
    signal PCI_X_Mode     : std_logic; -- PCI-Xモードレジスタ
    -- ↑PCI-Xモードなら'1'、PCIモードなら'0'(PCIバスリセット時は'0'とする)
    signal CLK_Mode       : std_logic_vector( 1 downto 0);
    -- ↑00=PCI:33MHz, 01=PCI/PCI-X:66MHz,
    -- 10=PCI-X:100MHz, 11=PCI-X:133MHz

    -- リセット信号保持フラグ. PCI/PCI-Xバスモード判定用
    signal    RST1 : std_logic;
    signal    RST0 : std_logic;

begin

-- ****************************************************** --
-- ***** PCI/PCI-Xモード判定
-- ****************************************************** --
process(PCICLK, nPCIRST)
begin
    if (PCICLK'event and PCICLK = '1') then
        if (RST0 = '0') then     -- PCIバスリセット中
            PCI_X_Mode     <= '0';
            CLK_Mode  <= "00";
            RST1      <= '0';
            RST0      <= '0';
        -- PCIバスリセットの立ち上がり検出
        elsif (RST1 = '0' and RST0 = '1') then
            if (nTRDY = '0' and nSTOP = '0') then
                PCI_X_Mode <= '1';          -- PCI-X 133MHz
                CLK_Mode   <= "11";
            elsif (nTRDY = '1' and nSTOP = '0') then
                PCI_X_Mode <= '1';          -- PCI-X 100MHz
                CLK_Mode   <= "10";
            elsif (nTRDY = '0' and nSTOP = '1') then
                PCI_X_Mode <= '1';          -- PCI-X 66MHz
                CLK_Mode   <= "01";
            else
                PCI_X_Mode <= '0';
                if ( M66EN = '1' ) then
                    CLK_Mode   <= "01";   -- PCI 66MHz
                else
                    CLK_Mode   <= "00";   -- PCI 33MHz
                end if ;
            end if;
        end if;
        RST0 <= nPCIRST;
        RST1 <= RST0;
    end if;
end process;
```

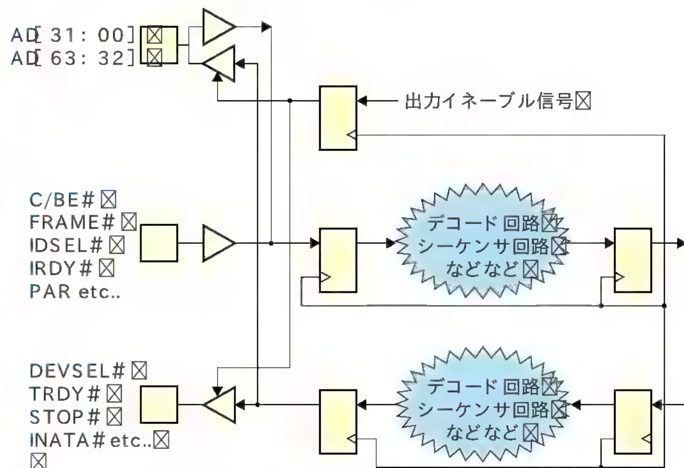
〔リスト 2〕信号同期化処理の例

```
-- ***** ADバス信号入力取り込み ***** --
process(PCICLK)
begin
    if (PCICLK'event and PCICLK = '1') then
        iAD <= AD;
    end if;
end process;

-- ***** DEVSEL_Ox/TRDY_Ox/STOP_Ox/
                        AD_OE/PAR_OD信号出力同期化 ***** --
process(PCICLK)
begin
    if (PCICLK'event and PCICLK = '1') then
        nDEVSEL_OD <= iDEVSEL_OD;
        nDEVSEL_OE <= iDEVSEL_OE;
        nTRDY_OD   <= iTRDY_OD;
        nTRDY_OE   <= iTRDY_OE;
        nSTOP_OD   <= iSTOP_OD;
        nSTOP_OE   <= iSTOP_OE;
        AD_OE      <= iAD_OE;
        PAR_OD     <= iPAR_OD;
    end if;
end process;

-- ***** ADバス信号出力同期化 ***** --
process(PCICLK)
begin
if (PCICLK'event and PCICLK = '1') then
        AD_OD <= iAD_OD;
    end if;
end process;
```

〔図 3〕入出力バスとのインターフェース図



PCI-Xのデバイスは，この図のようにすべての入出力ピンの値をI/Oパッドの直前でクロックに同期して，内部のステートマシンや外部出力を行う．トライステートバッファ信号群の出力制御信号も同様にクロック同期化して出力バッファ系の制御に用いる

のM66EN/STOP#/TRDY#の3信号の状態によって，クロック周波数とPCI-Xモードの検出を行っているというわけです．

● PCI-Xにおける各信号線の扱い

コンベンショナルPCIでは，シーケンサ内部が直接バス上の信号を参照し，そのときのイニシエータからの制御信号の状態に合わせてターゲットの信号を制御する必要があります．

しかしPCI-Xでは，シーケンサ内で直接バス上の信号を参照せずに，いったんデバイス内で各信号をクロックで同期化してから扱うようにします．アドレス/データバスとFRAME#/DEVSEL#などの各種制御線は，すべてこのルールに従って扱う必要があります．これはつまり，デバイス内部のシーケンサは，実際のPCI-Xバス上の動作より1クロック遅れてFRAME#やIRDY#などを判定し，さらに1クロック遅れてDEVSEL#やTRDY#を返すことになります．ロジアナなどで実際のバスの動きを観測するときには，この点に十分に注意してください．

例外的にデバイス内で同期化せずに直接参照している信号は，PCIRST#，PERR#，SERR#の3信号のみですが，これはコンベンショナルPCIバスでも同一です．

リスト2に制御信号同期化部を示します．また図3はPCI-Xのターゲットコントローラと外部入出力バスとのインターフェースを図に示したものです．この図でもわかるように，ターゲットデバイスはクロック同期による遅延などを考慮して設計する必要はありません．

逆に言えばコンベンショナルPCIバスではクロックと信号のスキューや配線遅延による遅れは絶えず考慮しなければならなかったのですが，PCI-Xではその分の制約がゆるやか

〔リスト3〕アドレスフェーズ/アトリビュートフェーズでの処理

```
-- ***** アドレスフェーズ信号生成 ***** --
Address_Phase <= '1' when (iFRAME_Delay = '1' and iFRAME = '0')
                                                        else '0';

process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        iFRAME_Delay <= '1';
    elsif (PCICLK'event and PCICLK = '1') then
        iFRAME_Delay <= iFRAME;
        -- ↑FRAME#信号を1クロック遅延させた信号を生成
    end if;
end process;

-- ***** アトリビュートフェーズ信号生成 ***** --
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        Attrbute_Phase <= '0';
    elsif (PCICLK'event and PCICLK = '1') then
        if (PCI_X_Mode = '1') then
            -- アトリビュートフェーズが存在するのはPCI-Xモード時のみ
            Attrbute_Phase <= Address_Phase;
            -- ↑アドレスフェーズの次のクロックがアトリビュートフェーズ
        end if;
    end if;
end process;

-- ***** データフェーズ信号生成 ***** --
process(PCICLK, nPCIRST)
    variable temp : std_logic;   -- テンポラリ
begin
    if (nPCIRST = '0') then
        Data_Phase <= '0';
        temp := '0';
    elsif (PCICLK'event and PCICLK = '1') then
        if (Data_Phase = '0') then  -- データフェーズ開始判定
            if (PCI_X_Mode = '1') then   -- PCI-Xモード時
                if (temp = '1') then
                    Data_Phase <= '1';
                end if;
                temp := Attrbute_Phase;
                -- PCI-Xはアトリビュートフェーズの次の次からデータフェーズ
            else                         -- PCIモード時
                if (Address_Phase = '1') then
                    Data_Phase <= '1';
                end if;
                -- PCIはアドレスフェーズの次からデータフェーズ
            end if;
        else              -- データフェーズ時終了判定(PCI/PCI-X共通)
            if (tget_HitDevice = '1') then  -- 自分が選択された場合
                if (iIRDY = '0' and iTRDY_OD = '0') then
                    Data_Phase <= '0';
                    -- ↑バースト転送非対応なので、
                    -- 最初のデータ転送が成立したら終了
                end if;
            else -- 自分が選択されなかった場合はすぐにクリア
                Data_Phase <= '0';
            end if;
        end if;
    end if;
end process;

-- ***** アドレスフェーズ時 アドレスフェーズ信号/コンフィグレーションサイク
                                            ル用IDSEL取り込み ***** --
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        tget_IDSEL <= '0';
    elsif (PCICLK'event and PCICLK = '1') then
        if (Address_Phase = '1') then     -- アドレスフェーズ検出
            tget_IDSEL <= iIDSEL;         -- IDSEL状態取得
        end if;
    end if;
end process;

-- ***** アドレスフェーズ時 アドレス取り込み ***** --
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        tget_Address <= (others => '0');
    elsif (PCICLK'event and PCICLK = '1') then
        if (Address_Phase = '1') then     -- アドレスフェーズ検出
            tget_Address <= iAD;          -- アドレス取得
        end if;
    end if;
end process;

-- ***** アドレスフェーズ時 バスコマンド取り込み ***** --
    tget_BusCommand <= iBusCommand;
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        iBusCommand <= (others => '0');
    elsif (PCICLK'event and PCICLK = '1') then
        if (Address_Phase = '1') then     -- アドレスフェーズ検出
            iBusCommand <= iC_nBE;        -- バスコマンド取得
        end if;
    end if;
end process;

-- ***** アトリビュートフェーズ時 アトリビュート取り込み ***** --
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        tget_Attrbute <= (others => '0');
    elsif (PCICLK'event and PCICLK = '1') then
        if (Attrbute_Phase = '1') then
                                      -- アトリビュートフェーズ検出
            tget_Attrbute <= iAD;     -- アトリビュート取得
        end if;
    end if;
end process;

-- ***** アトリビュートフェーズ時 バイトイネーブル取り込み ***** --
process(PCICLK, nPCIRST)
begin
    if (nPCIRST = '0') then
        attr_ByteEnable <= (others => '0');
    elsif (PCICLK'event and PCICLK = '1') then
        if (Attrbute_Phase = '1') then
                                  -- アトリビュートフェーズ検出
            attr_ByteEnable <= iC_nBE;
                                  -- シングル転送用バイトイネーブル取得
        end if;
    end if;
end process;
```

になったことで，クロック周波数を上げることができたといえるのです．

● アドレスフェーズとアトリビュートフェーズの識別

アドレスフェーズとアトリビュートフェーズでは，ADやC/BE#の信号を取り込みます(リスト3)．アドレスフェーズではADバスをアドレスとして，C/BE#をバスコマンドとして取り込みます．また，バスコマンドによってアトリビュートフェーズの意味合いが異なってきます．コンフィグレーションコマンドやメモリリードコマンドは，アトリビュートフェーズのC/BE#をバイトイネーブルとして取り込みます．メモリライトコマンドのバイトイネーブルはデータフェーズ中のC/BE#となる点に注意してください．

● PCI-X対応ターゲットシーケンサの動作

ターゲットシーケンサはコンベンショナルPCIもPCI-Xもほぼ同じですが，第2章で解説したように，制御線の制御ルールが異なる部分は，PCI-X用に変更する必要があります．リスト4

〔リスト 4〕PCI-X ターゲットのステートマシン

```
-- ********** BUS_IDLE 時の動作 ********** --
when BUS_IDLE =>   -- トランザクションの開始待ち
    if (iFRAME = '0' and iIRDY = '1') then -- トランザクション開始検出
         NSTATE := ADRS_COMPARE;
    else -- バスアイドル時このステートにとどまる
         NSTATE := BUS_IDLE;
    end if;

-- ********** ADRS_COMPARE 時の動作 ********** --
when ADRS_COMPARE =>   -- アドレスデコード結果を調べる
    if (tget_HitDevice = '1') then    -- 自分が選択された
         iDEVSEL_OD <= '0'; iDEVSEL_OE <= '1'; -- DEVLSEL# アサート
         iTRDY_OE <= '1';          -- TRDY# ドライブ開始
         iSTOP_OE <= '1';          -- STOP# ドライブ開始
         if (iBusCommand(0) = '0') then    -- リードサイクル時
              if (iIRDY = '0') then   -- イニシエータの準備完了
                   iAD_OE <= '1'; -- AD バスドライブ開始
              end if;
         end if;
         NSTATE := WAIT_IRDY;      -- イニシエータレディ待ちステートへ
    else -- 自分が選択されていない
         NSTATE := BUS_BUSY;       -- トランザクションの終了を待つステートへ
    end if;

-- ********** BUS_BUSY 時の動作 ********** --
when BUS_BUSY =>   -- トランザクション終了待ち
    if (iFRAME = '1') then  -- トランザクション終了へ
         NSTATE := BUS_IDLE; -- トランザクション開始待ちステートへ
    else -- トランザクション中ならこのステートにとどまる
         NSTATE := BUS_BUSY;
    end if;

-- ********** WAIT_IRDY 時の動作 ********** --
when WAIT_IRDY => -- イニシエータレディ待ち
    if (iIRDY = '0') then   -- イニシエータの準備完了
         if (iBusCommand(0) = '0') then    -- リードサイクル時
              iAD_OE <= '1'; -- AD バスドライブ開始
         end if;
         tget_AccStart <= '1';
         NSTATE := WAIT_LOCALRDY; -- ターゲットアクセス完了待ちステートへ
    else -- イニシエータの準備がまだならこのステートにとどまる
         NSTATE := WAIT_IRDY;
    end if;

-- ********** WAIT_LOCALRDY 時の動作 ********** --
when WAIT_LOCALRDY =>  -- ターゲットアクセス完了待ちステートへ
    tget_AccStart <= '0';
    if (PCI_X_Mode = '1') then        -- PCI-X モード時
         if (PCIX_TRDY_flg = '1') then  -- TRDY# アサート ok タイミング
              if (tget_AccReady = '1' or AccReady_flg = '1') then
                   iTRDY_OD <= '0';    -- TRDY# アサート
                   iSTOP_OD <= '0';    -- STOP# アサート
                   -- ↑シングルデータフェーズディスコネクト
                   NSTATE := ACC_COMPLETE; -- アクセス完了ステートへ
              else
                   NSTATE := WAIT_LOCALRDY;-- このステートにとどまる
              end if;
         else -- TRDY# をアサートできないタイミング
              if (tget_AccReady = '1') then     -- データ転送準備完了
                   -- 次のクロックで TRDY# アサートする
                        AccReady_flg := '1';
                   end if;
                   NSTATE := WAIT_LOCALRDY;-- このステートにとどまる
              end if;
    else                                  -- PCI モード時
         if (tget_AccReady = '1') then      -- データ転送準備完了
              iTRDY_OD <= '0';          -- TRDY# アサート
              NSTATE := ACC_COMPLETE;  -- アクセス完了ステートへ
         else
              NSTATE := WAIT_LOCALRDY;-- このステートにとどまる
         end if;
    end if;
    -- TRDY アサートタイミングフラグ反転
    PCIX_TRDY_flg := not PCIX_TRDY_flg;

-- ********** ACC_COMPLETE 時の動作 ********** --
when ACC_COMPLETE =>   -- アクセス完了ステート
    iTRDY_OD <= '1';                      -- TRDY# ディアサート
    if (PCI_X_Mode = '1') then            -- PCI-X モード時
         iDEVSEL_OD <= '1';               -- DEVSEL# ディアサート
         iSTOP_OD <= '1';                 -- STOP# ディアサート
         -- ↑シングルデータフェースディスコネクトの後処理)
         iAD_OE   <= '0';                 -- AD バスドライブ開放
         NSTATE := TURN_AROUND;           --  ターンアラウンドステートへ
         -- ↑ PCI-X モード時は FRAME# がディアサートされるのを確認する必要なし
    else                                  -- PCI モード時
         -- まだ FRAME# がアサートされている場合
         if (iFRAME = '0') then
              iSTOP_OD   <= '0';          -- STOP# アサート
              NSTATE := DIS_CONNECT;      -- ディスコネクトステートへ
         else                             -- シングル転送
              iAD_OE       <= '0';        -- AD バスドライブ開放
              iDEVSEL_OD <= '1';          -- DEVLSEL# ディアサート
              iSTOP_OD     <= '1';        -- STOP# ディアサート
              NSTATE := TURN_AROUND;      --  ターンアラウンドステートへ
         end if;
    end if;

-- ********** DIS_CONNECT 時の動作 ********** --
when DIS_CONNECT =>         -- ディスコネクト処理(PCI モード時のみ使われる)
    if (iFRAME = '1') then               -- イニシエータが STOP# を認識
         iAD_OE       <= '0';            -- AD バスドライブ開放
         iDEVSEL_OD <= '1';              -- DEVSEL# ディアサート
         iSTOP_OD     <= '1';            -- STOP# ディアサート
         NSTATE := TURN_AROUND;          -- 次は TURN_AROUND ステートへ
    else -- イニシエータが STOP# を認識していなければこのステートにとどまる
         NSTATE := DIS_CONNECT;
    end if;

-- ********** TURN_AROUND 時の動作 ********** --
when TURN_AROUND =>         -- ターンアラウンドステート
    PCIX_TRDY_flg := '1';
    AccReady_flg := '0';
    iDEVSEL_OE <= '0';               -- DEVSEL# ドライブ解放
    iTRDY_OE <= '0';                 -- TRDY# ドライブ解放
    iSTOP_OE <= '0';                 -- STOP# ドライブ解放
    NSTATE := BUS_IDLE;              -- トランザクション開始待ちステートへ
```

に PCI-X 対応ターゲットシーケンサを示します．

▶アイドルステート＆アドレスフェーズ待ち

リセット直後にステートマシンが動作開始するステートです．また，一連のバストランザクション完了後に戻ってくるステートもこのステートです．

このステートでは FRAME# のアサートと IRDY# のディアサートを認識してアドレスフェーズの開始を判断し，アドレス比較＆ DEVSEL 応答ステートに分岐します．またこのタイミングでリスト 3に示した回路でアドレスフェーズが検出されているので，AD バス上に出力されたアドレスと C/BE# に出力されたコマンドが取り込まれます．

▶アドレス比較＆ DEVSEL 応答

先のステートのタイミングで保持したアドレスと，内部ベースアドレスレジスタの値やコマンドと比較して，自身へのアクセスであるかどうかを判定するステートです．

アドレス比較の結果，そのトランザクションが自身へのアクセスであると判断した場合には，DEVSEL# をアサートして次のイニシエータレディ待ちステートに分岐します．ここでの DEVSEL# の出力はクロックで同期化されるので，実際の PCI-X バス上にはさらに次のクロックで出力されることに注意してください．そのため DEVSEL 応答は，Decode C のタイミングとなります．

そのトランザクションが自身ではない場合には，バスビジーステートに分岐します．

また，ちょうどこのステートを実行したタイミングで，リスト3に示した回路でアトリビュートフェーズが検出されているので，ADバスおよびC/BE#に出力されたアトリビュートが取り込まれます．

▶ バスビジーステート

このステートは，アドレス比較& DEVSEL応答ステートで自身のカードに対するトランザクションではないことを判断した場合に，そのトランザクションが完了するのを待つステートです．

本ステートではFRAME#がディアサートされるのを監視し続け，ディアサートされればトランザクションが完了したとみなし，アイドルステートに戻ります．

▶ イニシエータレディ待ちステート

イニシエータのデータ転送準備が完了したかどうかを判定するステートです．今回の設計は理解しやすさを念頭に置いたので，パイプライン動作などはさせていません．そこでイニシエータのデータ転送準備が完了したことを確認してから，バックエンドの回路を動作させるため，このステートが必要になります．

PCI-Xでは，アトリビュートフェーズの次の次のクロックには，必ずIRDY#をアサートするよう規定されているので，このステートの段階では必ずIRDY#がアサートされていることになります．ここではコンベンショナルPCIとの設計共通化のため，そのままにしています．

▶ ローカルレディ待ちステート

バックエンドに接続したRAMのアクセス完了待ちステートです．バックエンドに対するアクセススタート信号をクリアし，バスコマンドがリード系のコマンドの場合は，ADバスのドライブを開始します．

バックエンドからアクセス完了信号が返ってくれば1ワード分のアクセス完了なので，TRDY#をアサートします．また今回はシングル転送専用のターゲットなので，PCI-Xの場合はシングルデータフェーズディスコネクトでトランザクションを打ち切ります．シングルデータフェーズディスコネクトは，TRDY#のアサートと同時にSTOP#をアサートし，DEVSEL#をディアサートします．そして次にアクセス完了ステートに遷移します．

ここで，PCI-XとしてはTRDY#をアサートするタイミングに注意が必要です．PCI-XではTRDY#のアサートタイミングに関して，DEVSEL応答した次のクロックでアサートする場合をノーイニシャルウェイト，そこから2クロック後にアサートする場合を2イニシャルウェイトと呼び，以降は2クロックずつウェイトを増やせます．今回の設計ではDEVSEL応答の後，イニシエータレディ待ちステートを経てこのステートにきているので，最速でも2イニシャルウェイトになります．またTRDY#をアサートできるタイミングとできないタイミングを，1ビットのフラグを作ってそれをトグルしながら判定しています．

▶ アクセス完了ステート

直前のローカルレディ待ちステートでアサートしていたTRDY#やSTOP#などの信号をディアサートするステートです．またADバスのドライブもこのステートで開放します．

まず1ワード分のデータ転送が成立したので，TRDY#をディアサートします．またPCI-Xモード時はシングルデータフェーズディスコネクトのためにSTOP#もアサートしているので，ここでディアサートします．

コンベンショナルPCIのディスコネクトの場合，イニシエータがディスコネクトを認識するのを待つためのディスコネクトステートへ分岐しますが，PCI-Xのシングルデータフェーズディスコネクトの場合は，FRAME#がディアサートされるのをターゲットが待つ必要はありません．最後のステートであるターンアラウンドステートに分岐します．

▶ ディスコネクトステート

データフェーズステートで対応できないバースト転送要求が判断された場合にSTOP#をアサートしたのちに分岐するステートです．このステートではFRAME#がディアサートされるまでDEVSEL#とSTOP#をアサートし続け，イニシエータがディスコネクトを許諾するのを待ち続けます．イニシエータよりFRAME#がディアサートされた場合にはディスコネクト処理を終了するため，DEVSEL#とSTOP#をディアサートして，ターンアラウンドステートに分岐します．

アクセス完了ステートで説明したように，PCI-Xモード時はこのステートには遷移しません．

▶ ターンアラウンドステート

ターンアラウンドステートでは，いままでアサートしていたPCIADバスやDEVSEL#/TRDY#/STOP#の各信号の出力を開放してアイドルステートに復帰するステートです．各ステートから本ステートに遷移してきた時点で，DEVSEL#/TRDY#/STOP#の各信号はディアサート状態にあります．そこでこのステートで最終的に出力イネーブルバッファを無効にし，信号ドライブをハイインピーダンスにします．

その後アイドルステートに復帰して次のトランザクションの開始を監視します．

● コンフィグレーションレジスタについて

今回の設計で実装したコンフィグレーションレジスタを表7に示します．コンフィグレーションレジスタに関してはコンベンショナルPCIもPCI-Xも同一です．

設計したデバイス独自の仕様としては，現在のバスの動作モードやクロック周波数，そしてコンフィグレーションサイクル時のアドレスフェーズとアトリビュートフェーズの内容を保持したレジスタを，コンフィグレーションレジスタのオフセット+F0h以降のアドレスに配置しておきました．コンフィグレーションレジスタをダンプ表示することで，現在のモードやクロック周波数などを確認することができます．

〔表7〕コンフィグレーションレジスタ仕様

オフセット	レジスタ名称	値
PCI 標準仕様領域		
+00h	ベンダID	6809h
+02h	デバイスID	8000h
+04h	コマンドレジスタ	メモリ空間イネーブルビット
+06h	ステータスレジスタ	DEVSEL 応答 Decode C
+0Eh	ヘッダタイプ	01h(ヘッダタイプ1)
+10h	ベースアドレスレジスタ0	16Mバイトのメモリ空間を要求(32ビット/非プリフェッチ空間)
+2Ch	サブベンダID	6809h
+2Eh	サブシステムデバイスID	8000h
+3Ch	割り込みラインレジスタ	8ビット分実装(実際には未使用)
+3Dh	割り込みピンレジスタ	INTA# 使用(実際には未使用)
デバイス固有領域		
+F0h	リード/ライトテストレジスタ	32ビットリード/ライト可能レジスタ
+F4h	PCI-X モード/クロックモード	ビット31：PCI-X モード('1' で PCI-X) ビット1～0：クロックモード('11'：133MHz，'10'：100MHz，'01'：66MHz，'00'：33MHz)
+F8h	コンフィグレーションサイクルアドレスフェーズレジスタ	ビット15～11：デバイス番号 ビット10～8 ファンクション番号 ビット7～0：レジスタアドレス(実質F8h固定)
+FCh	コンフィグレーションサイクルアトリビュートフェーズレジスタ	ビット23～16：リクエスタバス番号 ビット15～11：リクエスタデバイス番号 ビット10～8：ファンクション番号 ビット7～0：セカンダリバス番号

上記以外は読み出し時'0'固定，書き込み時無視

4 デバイスに実装する際の注意点

ここでは，Stratix や Virtex-2/PRO の2種類の FPGA に，今回設計した PCI-X ターゲットシステムを実装する際の注意点などについて説明します．

● PLL/DCM を使用したクロック生成

最近の FPGA は非常に高速で駆動できるとはいえ，PCI-X の 100MHz や 133MHz で動作させるのはやはり難しい範囲になります．

カードエッジスロット経由で FPGA 内部にとりこまれたクロック信号をそのまま使用することになると，I/O パッド分の遅延時間(～3ns 程度)が発生してしまい，PCI-X の信号の入出力にも遅延時間が発生することになります．133MHz 駆動の場合には1サイクルが 8ns なので，I/O パッド分の遅延は無視できない値になってくるというわけです．

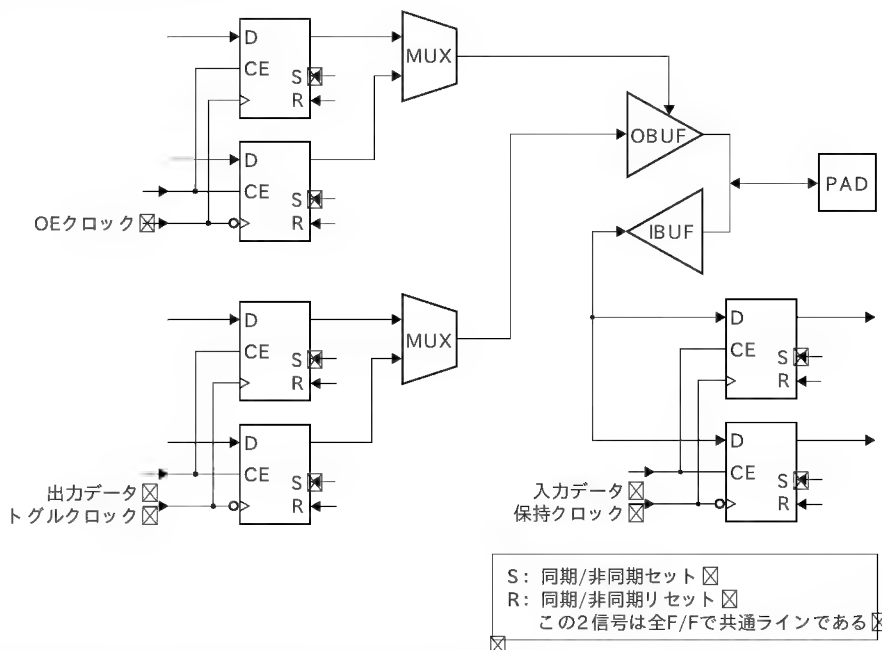
そこで，FPGA 内部の PLL や DCM を使って外側から供給されたクロックを同期化し，遅延時間分をキャンセルしたクロックで内部を動作させることで，デバイス外側の信号の動きとデバイス内側の回路動作を同位相で処理することができます．

ここで生成したクロックは，次に説明する I/O パッドレベルの信号同期や t_{SU}/t_{HOLD} 時間の確保に役に立ち，PCI-X を 100MHz や 133MHz で駆動させる場合には必須の技術です．

● 入出力信号の同期方法とピンレベルの t_{SU}/t_{HOLD} 確保の方法

PCI-X はバス上の信号をいったんクロックで同期化してから

〔図4〕入出力信号と直結した I/O パッドの構造



内部シーケンサなどで使用します．また出力信号側もデバイス内部で同期化した後に出力します．

このとき，同期化するためのレジスタ(フリップフロップ，以下F/F)をFPGA内部のロジックセルではなく，入出力ピンにもっとも近いI/Oパッド内部のレジスタに配置することで，次のような利点があります．

1) 入出力信号と直結したI/Oパッド内F/Fを使うことで t_{SU}/t_{HOLD} を確保することができる(図4)
2) FPGA内部のリソースを使わないので，ユーザーが使用できるロジック数が増える

ただし，同様にいくつか制限もあります．I/Oパッド内のF/Fは内部ロジックセルとは違い，セット/リセット周りやクロックイネーブルなど，サポートされていない機能があります．このため，ある条件で非同期にセット/リセットを行って信号を初期化するようなことはできないデバイスがほとんどです．そのため，I/Oパッド内のF/Fを使った設計は，あくまで信号入出力の間際で同期化するときにだけに使うのだということを念頭にいれておく必要があります．

今回使用したVirtex-2/PROは，同一の信号リソースであれば，I/Oパッド内のF/Fでも非同期のセット/リセット，また電源投入直後の信号の初期化さえも可能な構成になっています(図4中の注)．じつはPCI-Xのバスマスタ設計を行う際には，どうしても一部の設計で非同期的にセット/リセットを行いたいところがありまます．これについての具体的な解説は今回省きますが，PCI-X対応デバイスの設計を行う場合は，FPGA選定の目安としてI/Oパッドの構成にも気を配る必要があります．

Virtex-2/PROでI/Oパッド内のF/Fを使う場合に，Virtex-2/PRO専用のプリミティブライブラリを使用します．リスト5はVHDL言語で記述した例ですが，Verilogでも同様にmodule接続が可能です．これらのプリミティブライブラリは，設計ツールのメニューバーから「ヘルプ」→「オンラインヘルプ」→「ライブラリリスト」で確認することが可能です．

Stratixの場合もそうなのですが，最近の論理合成ツールと配置配線ツールは非常に賢くなっており，ソースコード内でリセットなどを入れないF/Fを記述することで，自動的にI/Oパッドに押し込まれ，I/Oパッド内のF/Fを使用してくれます(リスト5)．とくにアルテラの場合は，DDRメモリ用I/Oパッドを除けばユーザーが明示的にプリミティブライブラリを明記する必要はなく，自動的にI/Oパッド内のF/Fに配置されます．

● タイミングコンストレイン設定

最近のFPGAが高性能になったとしても，133MHzで駆動するPCI-Xコントローラを設計する場合には，コンストレイン(制約条件)を付加する必要性もでてくるでしょう．とはいえ，PCIやPCI-Xは規格化された標準バスなので，すでにデバイス設計ツールは標準でPCIおよびPCI-Xに対応しています．

ザイリンクスのデバイスの場合，I/O配置ピン定義と動作ク

〔リスト5〕入出力信号と直結したI/Oパッド内F/Fを使う場合のVHDL記述

```
-- ****************************************************** --
-- 入出力I/Oパッド部分の定義.
uMDQ_IOBUF1 : for i in 0 to 7 generate
    uMDQ_IOBUF2 : for j in 0 to 7 generate
         -- バスインターフェースは「SSTL2_II」を指定する
         uMDQ_IOBUF : IOBUF_SSTL2_II
              port map (
                   IO        => MDQ(i*8+j) ,
                   I         => MDQ_O(i*8+j) ,
                   O         => MDQ_I(i*8+j) ,
                   T         => MDQ_Tristate2(j)
              ) ;
    end generate ;
end generate ;

-- ****************************************************** --
-- DDR駆動出力レジスタの定義.
MDQ_Output_Equ1 : for i in 0 to 7 generate
    MDQ_Output_Equ2 : for j in 0 to 7 generate
         uMDQ_node : FDDRRSE
              -- セット/リセット付DDR-フリップフロップの呼び出し
              port map (
                   Q    => MDQ_O(i*8+j) ,
                   C0   => x1DDRCLK_90_I_n ,
                   C1   => x1DDRCLK_90_I_p ,
                   CE   => MDQ_OutShiftEN2(j) ,
                   D0   => phy_hld_MPUD_U(i*8+j) ,
                   D1   => phy_hld_MPUD_L(i*8+j) ,
                   R    => Always_L ,
                   S    => Always_L
              );
    end generate ;
end generate ;

-- ****************************************************** --
-- I/Oパッド内入力レジスタの定義
DDR_DQ_Equ_U1 : for j in 0 to 63 generate
    uDDR_DQ_Upper_Input_regs : FDRE
         port map (
              D    => dly_MDQ_I(j) ,
              C    => x1DDRCLK_90_I_n ,
              CE   => Always_H ,
              R    => Always_L ,
              Q    => hld_MDQ_O_U1(j)
         ) ;
end generate ;

DDR_DQ_Equ_L1 : for j in 0 to 63 generate
    uDDR_DQ_Lower_Input_regs : FDRE
         port map (
              D    => dly_MDQ_I(j) ,
              C    => x1DDRCLK_90_I_p ,
              CE   => Always_H ,
              R    => Always_L ,
              Q    => hld_MDQ_O_L1(j)
         ) ;
end generate ;

注意!!!!
 本当は，IFDDRRSEを使用して「明示的に入力I/Oパッド内DDRレジスタの使用」を行いたいのだがフィッタの制約により，FDDRRSEを出力側DDRレジスタとして使用している場合にはIFDDRRSEが使用できない(理由は不明．ほかにやり方があるのかもしれない... 誰かおしえて欲しい)
 ただし，FDDRRSEを出力信号系に，FDRE×2組を入力信号系に使用すると，自動的にI/Oレジスタ内の入出力側DDRレジスタを使用してくれる．

    component IFDDRRSE -- I/Oパッド内入力側DDRレジスタのコンポーネント
         port (
              D    : in STD_ULOGIC;
              C0   : in STD_ULOGIC;
              C1   : in STD_ULOGIC;
              CE   : in STD_ULOGIC;
              R    : in STD_ULOGIC;
              S    : in STD_ULOGIC;
              Q0   : out     STD_ULOGIC;
              Q1   : out     STD_ULOGIC
         );
    end component;
```

ロックのコンストレインを指定するには，UCFファイルを直接テキストエディタなどで編集するか，コンストレインエディタを使って記述します．リスト6はザイリンクス社のデバイス向けに，いくつかのPCIバスに関するインターフェースとタイミングコンストレインを記述したものです．PCI-Xの信号ピン名に「IOSTANDARD = PCIX」が記述されているのがわかります．

アルテラのデバイスの場合には，まずはじめにアサイメントエディタで「Use-PCI_IO＝ON」という設定を行ったほうがわかりやすいでしょう．というのはPCI用のI/Oを使うかどうかは，I/Oピン配置定義で使われるCSFファイルではなく，エンティティオプションを記述するESFファイルに設定されるからです．リスト7はESFファイルのコンストレイン情報を表したものです．

● I/Oバンクの使用方法

Stratix およびVirtex-2/PROともに，デバイス内部は八つのバンクに分かれており，それぞれのバンクごとにI/Oピンの信号電圧や電気的仕様を指定できます．

今回筆者が設計したPower-X/V2Pでは，DDRメモリとビデオインターフェース，Gbps差動インターフェース，そしてPCI-Xと，大別して四つのブロックに分かれています（図5）．そしてこのうちバンク6とバンク7をPCI-Xのインターフェースに割り付け，V_{CCIO}はPCI-Xカードエッジコネクタからの3.3V電圧を供給しています．

PCI-X/Mode1の信号電圧は3.3Vなのでこのような方法でも問題ありません，今回動作対応モードとしなかったPCI-X/Mode2の信号電圧は1.8Vとなるので，両方のモードに対応したPCI-Xアドインカードを設計する場合には，V_{CCIO}に供給するコア電圧の切り替えについても考慮する必要があります．

Virtex-2/PROの場合はいずれのI/OバンクでもPCI/PCI-Xに対応できますが，じつはStratixではPCI/PCI-Xとして使用可能なI/Oバンクは，八つのうちの2，3，6そして7だけなのです．Stratixを使ってPCIやPCI-Xアドインカードを設計する場合には，デバイスのピン配置決定段階で使用するI/Oバンクについて気をつける必要があります．

● Stratix評価キットでは

Stratix評価キットでは，32ビットPCIカードエッジ形状の拡張I/Oポートに接続されているバンクは，残念ながらPCI-Xを接続可能なバンクではありません．

とはいえ，実際問題として不都合が起きるかというと，そのようなことはまずありません．DC特性的にはPCI-XはLVCMOSレベルです．V_{oh}=0.9V_{CC}以上，V_{ol}=0.1V_{CC}未満となっており，スレッショルドレベルはテストコンディションにおいて0.6V_{CC}であるため，LVCMOS-3.3インターフェースでドライブすれば，とくに問題なく接続可能なのです（表8）．

またAC特性的にも，たしかにPCIバスの場合には－48mAなどという大電流駆動にも対応できるようにという記述があるのですが，マザーボードに実装されているプルアップ/プルダウン抵抗は数kΩ程度であり，デバイスの入出力容量/インピーダンスもCMOSであるため非常に高いという理由から，電流もほとんど流れません．

実際に筆者らは，Stratixに限らず，Cyclon（アルテラ）やSpartan-II（ザイリンクス）などのデバイスで，LVCMOS-3.3にして12mAのドライブ能力でPCIバスを駆動しています，不都合が生じたことはありません．

まとめ

PCI-XのトランザクションをPCI理解するためのPCI-Xターゲッ

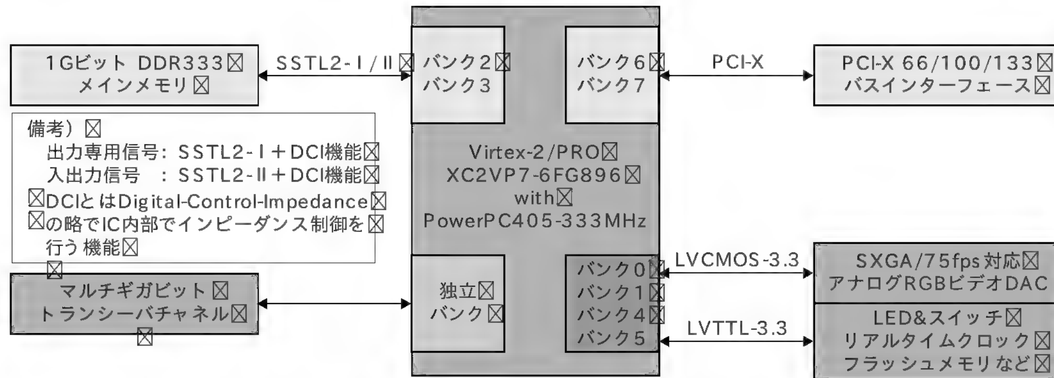
〔リスト6〕ザイリンクス/Virtex-2PROの場合の記述

```
# （　シャープはコメントを表す　）
# 入力PCIクロックの周波数規定を以下に行います．
# 1サイクル10nSec=100MHz，デューティー比1:1
#
NET "pciclk" TNM_NET = "pciclk";
TIMESPEC "TS_pciclk" = PERIOD "pciclk" 10 ns HIGH 50 %;
#
#PCIADバスのピン配置など・・・
NET "pciad<0>" LOC = "U23";
NET "pciad<1>" LOC = "U24";
NET "pciad<2>" LOC = "V25";
NET "pciad<3>" LOC = "V26";
#              |
#              |
NET "nframe" LOC = "P30";
NET "ndevsel" LOC = "R23";
NET "idsel" LOC = "U27";
NET "nirdy" LOC = "P29";
NET "ntrdy" LOC = "M28";
NET "nstop" LOC = "R25";
#
# PCI-X信号系のI/OインターフェースをPCI-Xに規定する．
# バスに関しては「*」をつかってワイルドカードで指定するのがスマート．
# Declare I/O-interface
NET "npcirst" IOSTANDARD = PCIX;
NET "pciad<*>" IOSTANDARD = PCIX;
NET "c_nbe<*>" IOSTANDARD = PCIX;
NET "nframe" IOSTANDARD = PCIX;
NET "ndevsel" IOSTANDARD = PCIX;
NET "idsel" IOSTANDARD = PCIX;
NET "nirdy" IOSTANDARD = PCIX;
NET "ntrdy" IOSTANDARD = PCIX;
NET "nstop" IOSTANDARD = PCIX;
NET "par" IOSTANDARD = PCIX;
NET "nperr" IOSTANDARD = PCIX;
NET "nserr" IOSTANDARD = PCIX;
NET "ninta" IOSTANDARD = PCIX;
NET "nintb" IOSTANDARD = PCIX;
NET "nintc" IOSTANDARD = PCIX;
NET "nintd" IOSTANDARD = PCIX;
#
```

〔リスト7〕アルテラ/Stratixの場合の記述

```
OPTIONS_FOR_NODES_AND_ENTITIES
{
    npcirst : PCI_IO = ON;
    c_nbe<0> : PCI_IO = ON;
    c_nbe<1> : PCI_IO = ON;
    c_nbe<2> : PCI_IO = ON;
    c_nbe<3> : PCI_IO = ON;
    nframe : PCI_IO = ON;
    ndevsel : PCI_IO = ON;
    idsel : PCI_IO = ON;
    nirdy : PCI_IO = ON;
    ntrdy : PCI_IO = ON;
}
```

〔図5〕I/Oバンク構成



〔表8〕PCI/PCI-Xの電気的特性（DCスペック）

記号	パラメータ	状況	PCI-Xバス規格		PCIバス規格（参考）		単位
			最小	最大	最小	最大	
V_{cc}	電源電圧		3.0	3.6	3.0	3.6	V
V_{ih}	HIGH入力電圧		$0.5V_{cc}$	$V_{cc}+0.5$	$0.5V_{cc}$	$V_{cc}+0.5$	V
V_{il}	LOW入力電圧		−0.5	$0.35V_{cc}$	−0.5	$0.3V_{cc}$	V
V_{ipu}	入力プルアップ電圧		$0.7V_{cc}$		$0.7V_{cc}$		V
I_{il}	入力リーク電流			±10		±10	μA
V_{oh}	HIGH出力電圧	$I_{out}=-500$〔μA〕	$0.9V_{cc}$		$0.9V_{cc}$		V
V_{ol}	LOW出力電圧	$I_{out}=1500$〔μA〕		$0.1V_{cc}$		$0.1V_{cc}$	V
C_{in}	入力ピン容量			8		10	pF
C_{clk}	クロックピン容量		5	8	5	12	pF
C_{IDSEL}	IDSELピンの容量			8		8	pF
L_{pin}	ピンレベルインダクタンス			15		20	nH

トコントローラの設計と，FPGAに落とし込む際のいくつかのノウハウや注意点について解説しました．

PCI-Xは，PCIと同一のラインがそのまま使用できるため，FPGAのようにあとから機能変更ができるようなデバイスの場合には，簡単にPCI-Xに切り替えができるという利点があります．今回Stratix評価キット向けに設計した回路では，PCIXCAPとM66ENピンをみてPCIバスとPCI-Xバスの上でどちらのモードであっても動作するターゲットコントローラを設計してみました．

バースト転送に非対応ですが，PCI-Xといえどもターゲット仕様であれば回路規模もそれほど大きくなく，それでいてクロック上昇分だけパフォーマンスが向上するので，FPGAを使って容易にアップグレードできるという利点と相まって，PCI-Xを使わない手はないと思うのですが，どうでしょうか．

参考文献
1) PCI-X Protocol Addendum to the PCI Local Bus Specification Revesion 2.0a，PSI-SIG.
2)「PCIバスの詳細と応用へのステップ」，『OPENDESIGN』，No.7.
3)「PCIデバイス設計入門」，『TECH I』，Vol.3，CQ出版（株）.
4)「量産時でも不ぐあいなく動くDDRメモリ・コントローラを設計する」，『DesignWave Magazine』，2003年11月号.

いくら・まさみ　来栖川電工有限会社

New Products ── IR，車載用インテリジェントパワースイッチ3種のサンプル出荷を開始
インターナショナル レクティファイアー ジャパン（株）は，高精度の電流検出機能をそなえ，電源電圧14V系の自動車用保護回路を内蔵したハイサイドIPS（インテリジェントパワースイッチ）IC，「IR3310/IR3311/IR3312」のサンプル出荷を開始した．

Appendix 01

ロジアナ波形で見るPCI-Xバスの動き

井倉 将実

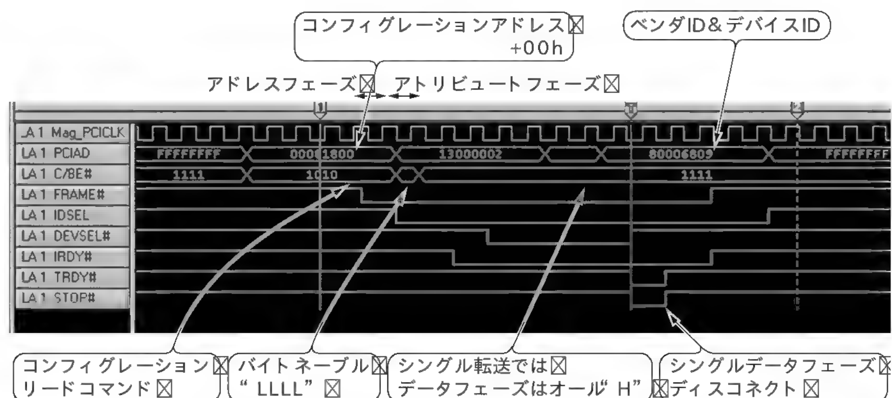
PCI-Xの各トランザクションの動作をロジアナで見てみましょう．特徴的なトランザクションを示すので，参考にしてください．

この波形は，ServerWorks製チップセットを搭載したマザーボードに，第3章で解説したStratix評価キットをPCI-X対応化したボードを実装した場合のものです．よって1ワードは32ビットとなります．たとえば図Dでは，一見4ワードバーストしているように見えますが，よく見ると2ワードバーストが2回連続で発行されているのがわかります．図Eはそれがより顕著にわかります．はじめに下位64ビット分の2ワードを転送したあと，少しアイドルを置いて上位64ビット分の2ワードの転送が行われています．また図Cのリードではリードブロックコマンドで実行されていますが，図Dのライトはライトブロックではなく通常のメモリライトコマンドでバースト転送が行われているのもわかります．

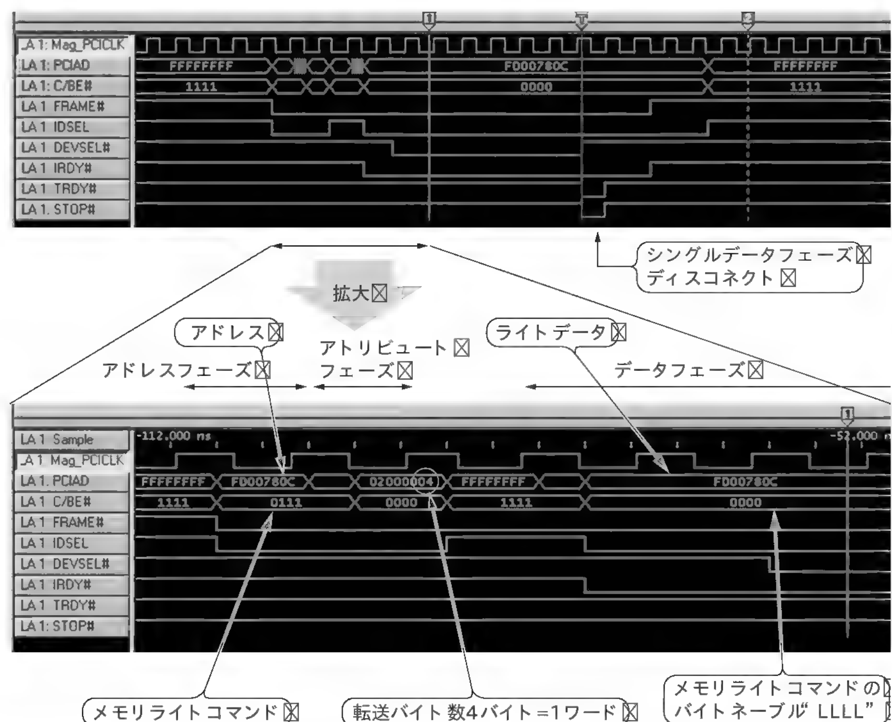
ちなみにIntel製チップセットの場合は，クワッドやダブルクワッドサイズでのアクセスが，また違った動作になるようです．

なお，クワッド/ダブルクワッドサイズでのPCIメモリ空間へのアクセスや，PCIメモリ空間をキャッシャブルに設定するプログラムについては，Appendix 3を参照してください．

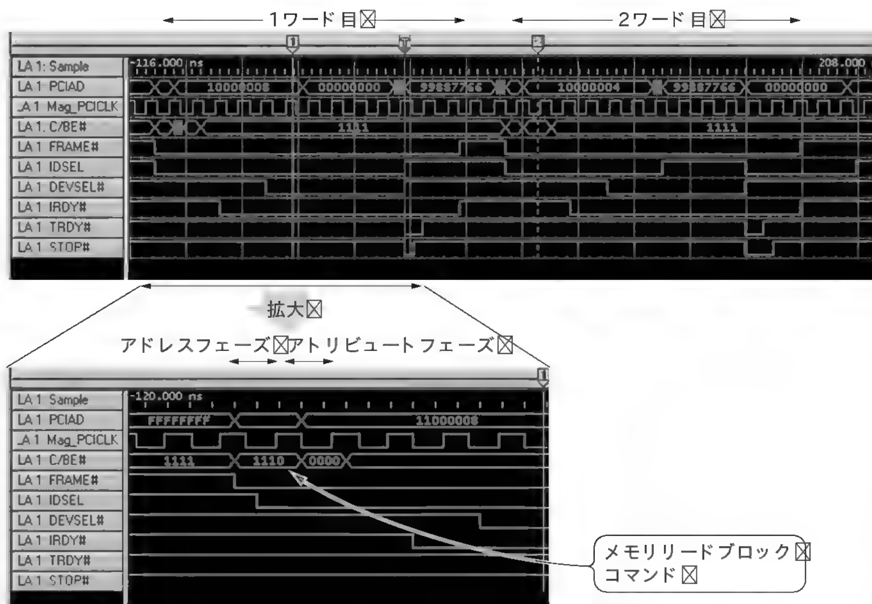
〔図A〕コンフィグレーションリードのようす



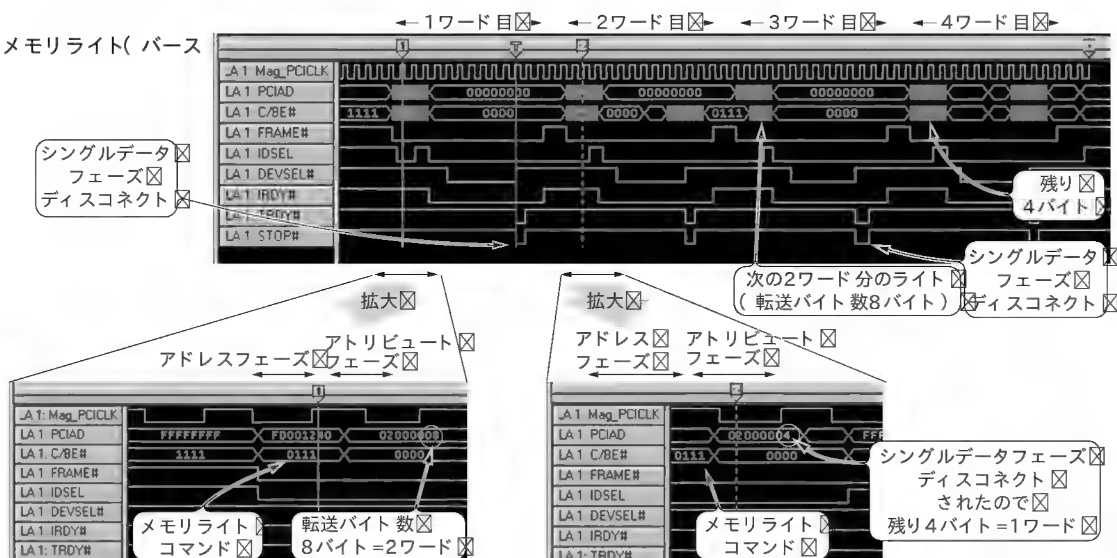
〔図B〕ロング(32ビット)メモリライト(シングル転送)のようす



〔図C〕
クワッド(64ビット)メモリリード(バースト転送)のようす



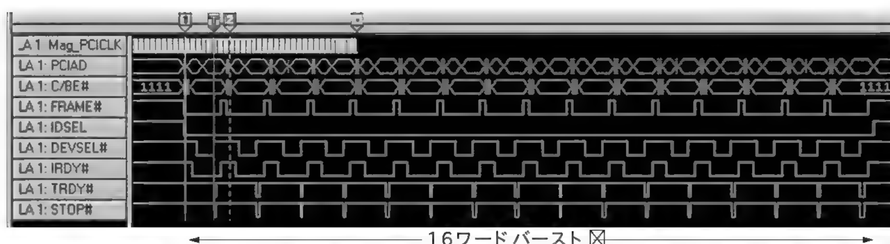
〔図D〕
ダブルクワッド(128ビット)メモリライト(バースト転送)のようす



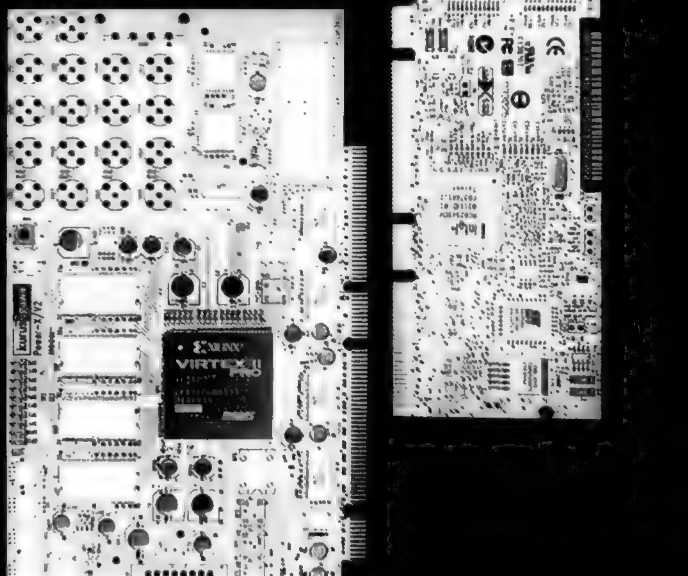
〔図E〕
ダブルクワッド(128ビット)メモリリード(バースト転送)のようす

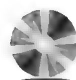


〔図F〕
メモリリードブロック(16ワードバースト転送)のようす



New Products ── NI, LabVIEW 音響/振動ツールセット 3.0 と次数解析ツールセット 2.0 を発売
日本ナショナルインスツルメンツ(株)は,「LabVIEW 音響/振動ツールセット 3.0」と「LabVIEW 次数解析ツールセット 2.0」を発売した.オーディオテスト,騒音/振動/ハーシネス(NVH)テスト,マシンコンディション監視などの音響振動解析アプリケーションの開発が行える.





バスマスタデバイスにユーザーメモリ空間をアクセスさせて高速化する

バスマスタPCIデバイス対応Windowsデバイスドライバの開発事例

山際 伸一

PCI-XはPCIと互換性があり，デバイスドライバについてもそれが適用できる．そこで，ここではバスマスタPCIデバイス用のデバイスドライバの作成事例を解説する．解説するドライバは，ユーザーメモリ空間にバスマスタPCIデバイスがアクセスできる領域を確保し，直接ユーザーアプリケーションとPCIデバイスとでデータの入出力を行えるというものである．

（編集部）

はじめに

PCI-Xデバイスは，ソフトウェア的には従来からのPCIデバイスとして見えます．したがって，PCIデバイスを制御するドライバについて説明すれば，それがそのままPCI-Xデバイスに対しても適用できます．とくに本章では，PCIデバイスがバスマスタとして動作しDMAコントローラをもっている場合に，ユーザーアプリケーション空間へ高速アクセスする方法について，ポイントをしぼって解説します．

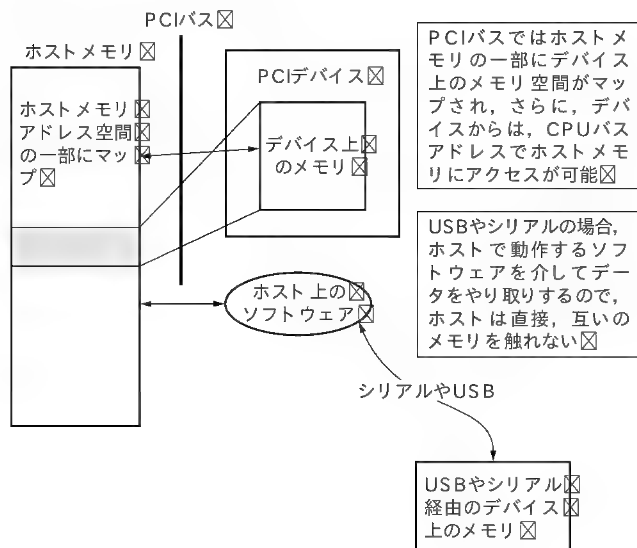
デバイスドライバについては，Windows2000およびXP向けのカーネルモードドライバをターゲットに話を進めます．INFファイルの書き方から，クラスインストーラ，および，ドライバ本体の記述方法について，説明していきます．PCIデバイスのドライバの場合，PnPマネージャによるリソース割り当てが自動的に行われるので，ある程度汎用的なカーネルモードドライバを作成できます．この機能についても言及します．

●PCIデバイスの利点

PCIデバイスは，USBやシリアル，パラレルなどと異なる性質をもっています．USBやシリアル，パラレルといった周辺バスには，デバイスのアドレス空間の一部またはすべてを，ホストマシンのメモリ空間に直接マップすることができません．また，デバイスがホストメモリに直接アクセスしてデータを読み書きすることは，その間のソフトウェアの介入なしには行うことはできません．

その反面，PCIバスに接続されたデバイスは，デバイスのアドレス空間の一部をホストメモリのアドレス空間にマップでき，さらに，ホストメモリに直接アクセスするバスマスタモードを選択できます（図1）．

〔図1〕PCIデバイスの利点



●ターゲットとなるドライバ

ここで作成するドライバはある程度，汎用性のあるものをめざします．これには，WindowsのPnP機能がリソース認識を行ってくれることを利用します．

ここで解説するドライバの機能の一つは，ユーザーアプリケーションで確保されたメモリをPCIデバイスに見せてあげる機能です．この機能を実現できると，アプリケーションとデバイスが直接協調動作でき，さらにDMAを使い高速なデータ転送が可能となります．

目標とするカーネルモードドライバの機能をまとめると，次のようになります．

（1）デバイスクラスの作成とPnP

デバイスクラスを作成しアイコンを設定できるようにします．これはINFファイルで設定できるので，この機能について説明します．アイコンを再登録するには，リソースファイルを編集できるツールが必要になります．手動で書き換えることもできますが，複雑になります．

（2）DeviceIoControlによるPCIデバイス空間へのアクセス機能

デバイスのリソースにアクセスできるようにします．これは，DMAコントローラなどのリソースにアクセスするために必要です．

（3）ユーザー空間のメモリ空間のページロック機能

PCIデバイスからのデータ転送をユーザー空間で受け取るにはいくつか方法がありますが，直接，ユーザーアプリケーションの動的な領域にアクセスをできるようにページロックをします．この機能については後で詳しく説明します．

● デバイス開発時の注意

デバイスからの直接アクセスを許すことで性能向上を見込める反面，開発時に危険がともなうことを肝に命じておく必要があります．デバイスはPCIバスを介し，直接ホストメモリを参照してきます．このときアドレスを間違っていると，たいへんな事態が発生します（筆者はこれで何度Linuxをぶち壊したことか……）．

Windowsの場合，デバイスがカーネル空間に誤って書き込みをしてしまった場合は，たいていの場合ブルースクリーンが現れます．心当たりのないエラーがブルースクリーンとして通知される場合は，たいていこのようなデバイスの不正アクセスが原因であると考えられます．DMAのアドレス設定の際は，このような「大惨事」を避けるよう，慎重にプログラムしてください．

1 Windows2000/XPカーネルモードドライバ入門

1.1 ドライバの役割について

● ドライバとは？

Windowsのユーザーアプリケーションを作ったことのある読者はわかると思いますが，アプリケーションが直接，デバイスメモリやI/Oにアクセスすることはできません．Windowsではプログラムの種類によって，デバイスリソースへのアクセスに関して権利が制限されています．つまり，ユーザーアプリケーションから，直接デバイスにアクセスするコードが書けないのです．これは，ユーザーが勝手にインストールしたプログラムにより，デバイスをガチャガチャと操作してシステムを壊すことも可能なので，このような保護機構が組まれているのです．Windowsのカーネル（具体的にはメモリマネージャ）によって，デバイスに割り当てられたシステムメモリアドレス空間やI/O空間は守られています．

では，ユーザーアプリケーションはどのようにしてデバイスにアクセスすればよいのでしょうか？　ユーザーアプリケーションへのアクセス許可をカーネルとの間で「お話」してもらう何かを用意する必要があります．その何かがデバイスドライバということになります．したがって，デバイスドライバは，ユーザーアプリケーションからの要求をカーネルに橋渡しすることに加え，ユーザーアプリケーションにアクセスの可/不可を通知する仲介役を担うことになります．さらに，この仲介役は通訳もします．ここでいう通訳とは，アプリケーションへのインターフェースを標準化し，デバイス個別のアクセス方法を隠ぺいするということです．

● ドライバ動作モードによる分類

Windowsでは，デバイスドライバの動作モードとして，大きく二つの分類があります．一つはカーネルと同一のモードで動作するカーネルモードドライバで，もう一つはユーザーアプリケーションと同一のモードで動作するユーザーモードドライバです．カーネルモードドライバは，デバイスのリソースに直接アクセスできますが，ユーザーモードドライバは，Win32 APIを使ってアクセスする必要があります．

ユーザーモードドライバの例としては，プリンタドライバがあります．昔のプリンタはLPTポートに接続するものでしたが，最近ではEthernetやUSBなど，LPT以外のインターフェースで接続することも多くなりました．そこでプリンタドライバは，物理的な接続インターフェースに囚われずに，プリンタに対する出力データを用意する部分のみを担当するように，役割が分担されてきています．この場合，物理的なハードウェアには直接アクセスしないので，ユーザーモードドライバとして定義されているのです．

● カーネルモードドライバとは？

一方，カーネルモードドライバでは，「何でもあり」の環境下で動作します．つまり，デバイスにアクセスするための空間に，メモリマネージャの仲介なく直接アクセスすることが可能です．ここでは，このカーネルモードドライバに焦点を当てて解説します．

カーネルモードドライバは，標準的な機能に分類されるデバイス（標準デバイスクラス）と，それらに分類されないデバイスクラス（インターフェースクラス）に分類されます．標準デバイスクラスは，たとえばビデオカード，サウンド，ネットワーク，マウス，キーボードといった，すでにインターフェースが決まりきっているものに関して，Windowsが専用のデバイスクラスを用意しています．これらのタイプのデバイスドライバを作成する際には，Windowsの用意するリソースをそのまま用いるのがもっとも簡単で，信頼性のあるものが作成できます．さらに，ドライバインストールの際にはWindowsがデバイスクラス固有のインストール手順を用意しているので，ドライバ開発時の手間を最小限にすることができます．

一方，インターフェースクラスは，開発者が標準的なデバイスと異なる方法でアクセスすることを前提としたアプリケーションを対象としたクラスです．このデバイスクラスは，クラスのアイコンやクラス名まですべて開発者側で定義可能ですが，設定方法が事前に用意されていないので，クラスインストーラと呼ばれる，インターフェースクラスをインストールするためのルーチンを用意する必要があります．クラスインストーラはDLLとして用意されます．このDLLには，デバイスクラスア

イコンも含まれます．

DeviceIoControlで自由にデバイスにアクセスすることを前提とする場合には，インターフェースクラスを選択する必要があります．ここでは，このインターフェースクラスを用いてDMAコントローラを制御するドライバを作成していきます．

以上のデバイスの分類に加え，Windowsでは，WDM（Windows Driver Model）と呼ばれるドライバの作成方法が推奨されています．本章では，このWDMに沿って設計を進めていきます．WDMとは，プラットホーム依存でないドライバの階層やアクセス方法，要求の発行方法などを定義したものです．

● ドライバの階層別分類

Windowsのドライバは階層別に分類すると，Filter Driver，Function Driver，Bus Driverの三つがあります（図2）．

Filter Driverは，その上位または下位に渡す要求を加工するドライバです．通信プロトコルや，イメージフィルタなどが該当します（厳密に分類すれば，ファイルシステムもFilter Driverに分類される）．

Function Driverは，デバイスを制御するためのドライバです．通常，PCIデバイスドライバといった場合，Function Driverを指します．

Bus DriverとはPCIやUSBなどの物理的なバスを制御するためのドライバです．PnPマネージャがリソースを要求する際には，この階層に要求を出します．このドライバは通常，マイクロソフトによって提供されているので，デバイスが既存の標準的なバスを用いている場合は新たに追加する必要はありません．PCIにもPCIバスドライバがデフォルトでインストールされています．

この階層の上にはサブシステム（通常ファイルシステム）が存在し，アプリケーションソフトウェアの要求をカーネル内部のデータ構造に変え，さらに適当と思われるデバイスクラスに渡してきます．このときの要求のことをIRP（I/O request packets）と呼びます．IRPは上位階層から順に渡されていき，処理すべきドライバの階層が処理します．

WindowsではこのようにアプリケーションからIRPを階層にしたがって渡していくことで，デバイスへのI/Oを要求します．IRPが処理されるとサブシステムへと返され，アプリケーションに要求の返答内容が返されます．

1.2 インストールからアプリケーションまで

ドライバの作成を始める前に，どのような流れでデバイスがインストールされ，システムに常駐し，アプリケーションからアクセスされるようになるのか，全体のストーリがわかっていると，プログラムを書く際に詳細を理解する手助けとなります．

では，ドライバがインストールされるところから，アプリケーションでデバイスアクセス要求が発生されるまでの流れについて見てみましょう．ここでの説明対象はインターフェースクラスのPCIデバイスドライバとします．

● 新しいデバイスの発見

新しいデバイスがPnPマネージャにより発見されると，デバイスのインストールウィザードが呼び出され，ドライバのインストーラが開始されます．まず，INFファイルを探すところから手順は始まります．INFファイルには，デバイスのインス

Column 1

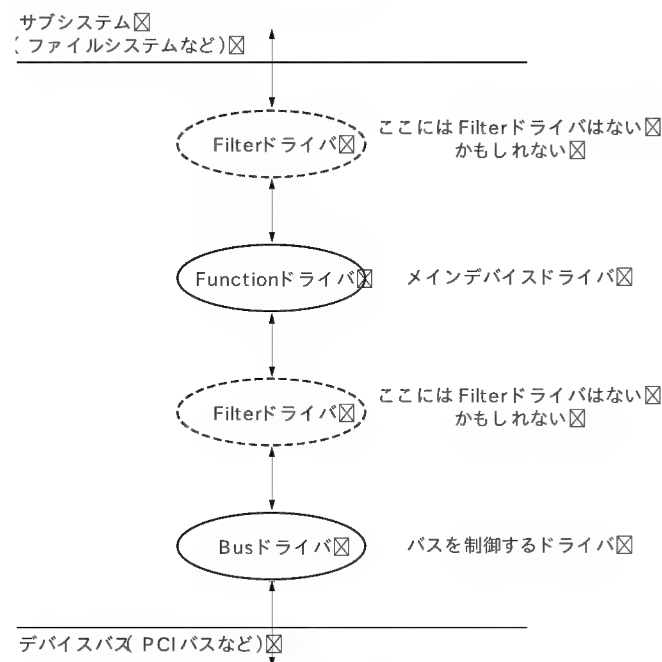
Windowsにはmmap関数がない!?

LinuxなどのUnix系OSのドライバ開発をしていた方が，すでにあるLinuxドライバをWindowsのドライバへ移植しようとして，戸惑う部分の一つに「mmap関数がWindowsにはないのか？」があります．そうです，Windowsにはないのです．

すべての基本アクセスはDeviceIoControlで行われます．これを駆使してデバイス空間へアクセスするアルゴリズムを考えなければなりません．つまり，mmap関数によりアプリケーションレベルでデバイスのアドレス空間をマップしている場合には，ポインタで操作していた部分をすべて変更することになります．

LinuxとWindowsを両方サポートする環境を開発する場合，mmap関数の使用を避けると移植性が良くなりますが，デバイスリソースを直接，ユーザー空間にマップできないため，アクセススピードは下がります．このトレードオフを考えるべきでしょう．

〔図2〕Windowsドライバの階層別分類



トールに関する情報が書かれています．デバイスインストーラは，このINFファイルを設計図にして，クラス，ドライバの順にインストールを行っていきます．

PnPマネージャがINFファイルを発見すると，INFファイルの中にあるデバイスリストを探します．デバイスリストはManufactureセクションに書かれています．このリストを読んで，インストールウィザードに一覧を表示しています．

● クラスイントーラ

デバイスを選択すると，インストール作業が開始されます．このとき，PnPマネージャによってデバイスクラスがチェックされます．[Version]セクションの「Class = ??」と書かれている行のデバイスクラス名をチェックし，標準デバイスクラスかどうかを調べます．標準デバイスクラスの場合（たとえばNetなど），そのクラスに該当するクラスインストーラを呼び出し，インストール作業が続行されます．

ここでの例では，インターフェースデバイスクラスなので，Windowsにデフォルトでは存在しないクラスです．このような場合，PnPマネージャはクラスインストーラに関するセクション「ClassInstall32」を探し，DLLとして提供されるクラスインストーラのインストールを進めます．このとき，PnPマネージャはクラスをレジストリに登録しますが，その際にGUID（コラム2参照）と呼ばれるクラスにユニークなIDをキーにして登録を進めます．

● デバイスドライバのインストール

その後，デバイスドライバのインストールが開始されます．デバイスにはモデルによって動作が異なるものがあり，バージョンが異なるだけでも動作が異なるものがあります．このような差異に対応するために，INFファイルではハードウェアIDを記述することで，ドライバが対応するデバイスかどうかの一致を図ります．PCIデバイスの場合，ハードウェアIDは以下のように指定します．

```
PCI¥VEN_ベンダID&DEV_デバイスID&SUBSYS
                    _サブシステムID&REV_リビジョンID
```

この最初のPCIというのが，エニュメレータと呼ばれるものです．エニュメレータはPnPマネージャによりハードウェアIDを元に指定されます．PCIエニュメレータはクラスインストーラと協調しながらPCIデバイスドライバとしてインストールを進めていきます．PCIエニュメレータはドライバのエントリポイントを指定し，さらにドライバへデバイスリソースを渡し，システム（レジストリ）にデバイスを登録します．ここまでくると，デバイスがシステムにインストールされ，アプリケーションから利用可能になります．

● デバイスのオープン

アプリケーションでは，デバイスのハンドルを獲得する処理から始まります．つまり，デバイスをオープンします．このとき，GUIDを元にデバイススペシャルファイルへのパスを獲得する必要があります．デバイスへのファイルパスを獲得し，

Column 2

GUIDって何？

Windowsでは，デバイスはスペシャルファイルとして定義されていますが，そのファイル名を探索する際に，GUIDと呼ばれる設計者側が決定した番号を元に探索します．GUIDは，クラスに対しても定義する必要があります．

では，なぜGUIDが必要なのでしょうか？　これは，Windowsの世代が変わっても，GUIDでのファイルまたはクラスの探索が保障されるようになっているためです．Windowsのバージョンによって，レジストリやスペシャルファイルパスが異なる場合があります．このような場合，GUIDを使い機械的に探索対象を一致させる必要があるためです．

GUIDはドライバ開発時点でのシステム時刻と開発マシンのID（MACアドレスやHDDのシリアルナンバ）を元に作成されます．GUIDは，DDKに付属する`guidgen.exe`を用いることで簡単に作成できます．

`CreateFile`関数でファイルを開くと，デバイスへアクセス可能な状態になります．

ここで，デバイスへアクセスする例として，`DeviceIoControl`関数を用いた場合の例を説明します．アプリケーションから`DeviceIoControl`が呼ばれると，ファイルシステムを介し，入出力バッファの情報がIRPに格納され，このIRPがデバイスドライバに渡されます．

この際，IRPの要求に一致するルーチンがドライバの中から選択され（ディスパッチルーチンと呼ばれる），実行されます．アプリケーションに指定されたI/Oコントロールコードに一致する処理が行われ，IRPの完了がアプリケーションに通知されます．

このとき，`DeviceIoControl`関数の完了と，IRPの完了は必ずしも一致していないことに注意してください．`DeviceIoControl`関数の完了は，ディスパッチルーチンの完了に同期しますが，IRPの完了はそれとは別になります．このようなアクセスのタイプを非同期I/Oと呼びます．非同期I/Oは入出力に必要な時間が長いデータアクセスに対して実装されます．ファイルシステムは完了したIRPに格納されている入出力データを展開し，アプリケーションへと返します．

1.3　カーネルモードドライバの構成

それでは，ドライバ本体の中身がどのようになっているか見てみましょう．カーネルモードドライバはある程度フォーマットが決まっているので，それにしたがって作ることで，実装することができます．

IRPの種類に一致する処理ルーチンのことをディスパッチルーチンと呼びます．処理フローを元に次のようにディスパッ

Prologue
Chapter 01
Chapter 02
Chapter 03
Appendix 01
Chapter 04
Chapter 05
Appendix 02
Chapter 06
Appendix 03

Column 3

IRQLって何？

DDKドキュメントの関数の説明を見ていると，

IRQL >= DISPATCH_LEVEL

のような記述を見つけると思います．この記述は関数の実行可能レベルを表しています．

カーネル内で用いられる関数には実行される優先順位があり，そのレベルをIRQレベルとWindowsでは呼んでいます．このレベルの条件に合わないと，関数が実行されなかったり，不正なアクセスが発生したりして，正しく実行されないようになっています．

チルーチンをまとめました．それぞれのルーチンを説明していきます．

● DriverEntry

このルーチンは，エニュメレータがドライバをインストールする際に最初に実行されるルーチンです（エントリポイントと呼ばれる）．このルーチンでは，ドライバオブジェクト構造体（この構造体はドライバ情報をアンロードされるまで保持する）にディスパッチルーチンを登録します．

● AddDevice，PnPディスパッチルーチン

DriverEntryの次に実行されるルーチンです．このルーチンではデバイスのリソースがIRPにセットされ，送られてきます．そのリソースをどのようにマップするかを記述します．

● IoControlディスパッチルーチン

このルーチンはユーザーアプリケーションでDeviceIoControl関数が実行されたときに実行されます．DeviceIoControl関数では，I/O要求の種類，入力バッファとそのサイズ，出力バッファとそのサイズ，および，ユーザー空間とカーネル空間との間でそれらの受け渡しをどのようにやるかが記述されます．このとき，デバイスドライバにはそれらのポインタまたは，データのコピーがIRP内部に保存され，渡されてきます．

このルーチンの中で，I/O要求の種類を判別し，入出力バッファのデータを処理します．また，I/Oが完了できる場合はIRPを完了させますが，そうでない場合は，IRPをペンディング状態にして返ります．

● Unloadディスパッチルーチン

デバイスドライバがアンインストールされるときに呼ばれます．このルーチンが呼ばれるとき，ドライバ内で用いられていたメモリ領域を開放することがおもな仕事になります．

● Createディスパッチルーチン

このルーチンはCreateFile関数が呼ばれた際に実行されます．このルーチンでは，とくに行う処理はありません．渡されたIRPを完了させるだけになります．

● Close，Cleanupディスパッチルーチン

デバイスハンドルが閉じられるとき，プロセスが終了する際に呼ばれます．ドライバハンドルを保持するプロセスが終了する際には，Cleanup→Closeの順でIRPを発行します．このルーチンでは掃除をしてIRPを完了します．

1.4 開発環境（DDK）について

● 開発言語

ドライバの開発はC言語とアセンブラで行います．基本的には，DDKの提供するマクロや，関数をドキュメントに指示されたとおりに記述していくと，簡単なドライバならすぐにできてしまうというしくみになっています．

● DDKを入手する

カーネルモードデバイスドライバを作るには，DDKの入手が不可欠です．最新のWindows XP DDKは，次のようなパッケージになっています．

1) DDKドキュメント
2) WIndows2000，XP，Me，および，VxD向けライブラリとインクルードファイル
3) Build環境（build，nmake，Cコンパイラなど）
4) ツール（guidgen，Infgen，Infchkなど）

現在の配布方法は，マイクロソフトからのCD-ROM配布（Webサイトから要求すると無料で取得できる），または，MSDNサブスクリプションからのダウンロードが可能です．

http://www.microsoft.com/whdc/ddk/winddk.mspx

MSDNサブスクリプションは，最新のWindows環境を含む，開発者専用ライセンスを提供しています．サブスクリプションレベルにより，Visual Studioなどのアプリケーション開発ツールを含む，多くの開発資料を取得することができます．これから，本腰をあげてドライバ開発を行おうとしている読者には，ぜひ加入することをお勧めします．個人で加入するには，MSDNパッケージ製品を店頭で購入するのがもっとも手ごろだと思います．

MSDNサブスクリプションでの大きな利点は，チェックドビルドWindowsが手に入るというところではないかと思います．後述しますが，カーネルモードドライバ開発の際には，たいへん役に立ちます．

● デバッグ方法

▶ チェックドビルドWindows

通常店頭販売されている一般向けのWindowsはフリービルドWindowsと呼ばれ，最高速の性能を出すようにビルドされたものです．それに対して開発者向けWinwowsが存在します．それがチェックドビルドWindowsです．チェックドビルドWindowsは，カーネルの起動メッセージが出力されるだけでなく，カーネルモードデバイスドライバのメッセージを見ることができます．

〔表1〕チェックドビルドWindowsでサポートされるおもなデバッグ用ルーチン

関数名	機　能
DbgPrint()	printfライクな引き数をとり，メッセージを出力する．変数などを参照できる
DbgBreakPoint()	ブレークポイントを設定する
ASSERT()	引き数の条件文が不成立の場合ブレークが発生する

チェックドビルドWindowsでのみサポートされるおもなデバッグ用ルーチンを表1に示します．これらのルーチンを駆使して，カーネルモードドライバの中で何が起こっているかを観察しながら開発を進めることができます．

ASSERT関数に関しては，フリービルドWindowsのカーネルモードドライバ用の関数にも存在しています．よって，異常な引き数で関数を呼んだときにもASSERTでひっかかることがあります．

▶WinDbgデバッガ

どのようにチェックドビルドWindowsからの出力を見るかですが，マイクロソフトからデバッガが提供されているので，それを用いるのがもっとも手ごろな手段です．

Windows用デバッガWinDbgは次のURLからダウンロード可能です．

http://www.microsoft.com/whdc/ddk/debugging/default.mspx

WinDbgを用いるときは，図3に示すように，デバッグ対象のターゲットマシンとデバッガが動作するホストマシンをシリアルクロスケーブルで接続した構成をとる必要があります．またリスト1のように，ターゲットマシンのWindowsのブート時の設定(デフォルトだとC:¥boot.iniファイル)を変更します．リスト1の例は，Windows2000のデバッグモードを有効にし，デバッグ出力をシリアルケーブルにボーレート115200bpsで接続する設定です．

● ビルドの方法

ビルドの際に必要な設定は，makefileとsourcesファイルを用意することです．この二つのファイルはbuildコマンドにより解析され，自動的にコンパイル後の実行形式とソースファイルのコンパイル方法が決定されます．

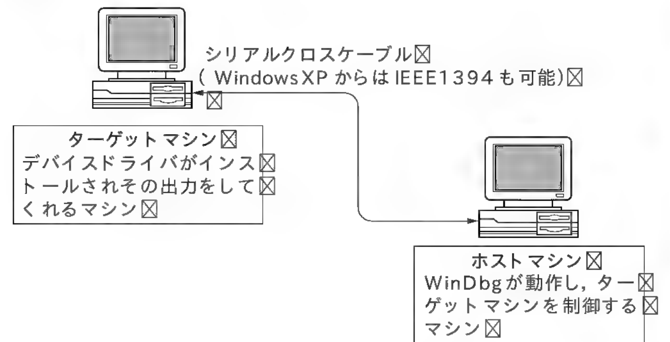
makefileに関しては，次の1行以外は書かないでください．この設定は，DDK用の環境が設定されています．

```
!INCLUDE $(NTMAKEENV)¥makefile.def
```

コンパイルされるソースファイルや，コンパイル後の形式の指定に関しては，sourcesファイルに記述します．リスト2に後述するPCIDMACドライバのsourcesファイルを示します．

TARGETPATHはオブジェクトファイルの情報を書き込むフォルダパスを指定します(必須)．TARGETTYPEはDRIVER，DLL，PROGRAMなどの種類を指定します．それぞれ，sys，dll，exeなどの拡張子がTARGETNAMEで指定されたコンパイル後の名前に付加されます．LINKER_FLAGSにはリンカへのデフォルト設定への追加フラグを指定します．-MAPはアドレスマップファイルを出すという意味です．SOURCESにはターゲットを構成するソースファイルを指定します．

〔図3〕デバッグ環境



〔リスト1〕ターゲットマシンのWindowsブート設定

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)¥WINNT
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)¥WINNT=
"Microsoft Windows 2000 Professional English (Checked/Debug)" /
debug /debugport=COM1 /boudrate=115200
```

〔リスト2〕作成したPCIDMACドライバのsourcesファイル

```
TARGETPATH=obj
TARGETTYPE=DRIVER
TARGETNAME=DMAC_drv

LINKER_FLAGS=-MAP

SOURCES= DMAC_drv.c
```

以上，二つのファイルを用意したあと，[スタート]→[プログラム]→[Development kits]→[Windows DDK]→[Build Environments]→[Win 2K Free Build Environment]を開きます．そして，それらがあるフォルダにカレントディレクトリを移動します．その後，buildコマンドを実行すると一連のコンパイル作業が始まります．

WindowsXP DDKを用いている場合，フリービルド向けのコンパイル結果はobjfre_w2k_x86¥i386フォルダに保存されます．

2 Windowsでユーザーアプリケーションのメモリをピンダウンする方法

さて，ここまでで，Windows2000以降のドライバの作成の基礎はおさえられたと思います．ここからは，PCIデバイスのDMAコントローラを制御する方法について，必要な知識とともに，実際のコーディングを見ていきましょう．

2.1 メモリピンダウン

● メモリマネージャの役割

“ピンダウン”という言葉で連想されることはなんでしょうか？　直訳すると「ピンで対象となるものを留める」ということです．つまり，その場から物を動かないようにくくり付けるということです．バスマスタPCIデバイスのDMAコントローラがホストメモリを直接触る場合，メモリをピンダウンする必要があります．それはなぜでしょうか？

Windowsを含め，オペレーティングシステムには，メモリマネージャと呼ばれる物理メモリを管理する機能が実装されています．メモリマネージャはメモリに関するすべての複雑な処理を行うわけですが，おもな処理としては次のようなものがあげられます．

(1) アドレス管理

プロセスが起動する際や，アプリケーションの中でmalloc関数などを使って新たなメモリ領域を確保する場合，メモリマネージャにその処理をまかせることになります．メモリマネジャは物理メモリ空間の一部分を新たに確保する領域に割り当てるわけですが，その際に，仮想アドレスに物理メモリをマップするアドレス変換テーブルを作成します．

これは，ほかのプロセスからその領域を守るということと，仮想的に巨大なメモリ空間(32ビットCPUの場合4Gバイト)を実現するという機能を実現しています．

(2) ページング

仮想的に大きなメモリ空間を仮想アドレスによって実現できても，実際はそのような物理メモリは存在しない場合があります．しかし，OSは動作しなければなりません．この問題に対処しているのが，ページングと呼ばれる機能です．

ページングは，動作頻度の低いプロセスに関するメモリ領域や，実行中のプロセスの領域の中で，使用されていないメモリ領域を見つけ，ディスクに一時的に退避し(スワップアウトと呼ぶ)，新たなメモリ確保要求に対応できるようにします．

退避されたメモリ領域が再度，必要になった場合には，ディスクに書き戻した情報をメモリに戻します(スワップインと呼ぶ)．このときの領域は，退避前の場所とは限りません．そこで，仮想アドレステーブルも書き換えられます．メモリマネージャは，スワップイン/スワップアウトを繰り返し，多くのプロセスを一気に実行できる環境を作り上げています．

このスワップはページと呼ばれる単位で行われています．ページとは，OSの定義する物理メモリのブロックのことです．i386アーキテクチャで動作するWindows2000またはXPの場合，4Kバイト(コラム5参照)と定義されています．

(3) ページ属性の管理

メモリマネジャは，物理メモリページに関しての属性も管理しています．ここでいう属性とは，キャッシュの有無です．CPUのキャッシュを用いて，その領域をアクセスするのか，すべてのCPUアクセスがキャッシュをバイパスし，メモリへのアクセスを行うべきなのかを設定します．

● PCIデバイスからの物理メモリアクセスの際の問題点

ここで，バスマスタPCIデバイスのDMAコントローラが，物理メモリに直接アクセスする場合の問題点を挙げてみます．

(1) プロセスの仮想アドレスには直接アクセスできない

PCIデバイスの扱うアドレスは，CPUの外部バスがアクセスするのと同じ物理アドレスです．しかし，プロセスは仮想アドレスを元に動作しています．仮想アドレスは，必ずしも物理アドレスと同一であるとは限りません．そこで，PCIデバイスが直接プロセスのメモリにアクセスする場合には，何らかの方法で仮想アドレスに一致する物理アドレスを取得する必要があります．

(2) 物理メモリページがスワップアウトされる

ページはメモリマネージャによって，スワップアウトされてしまうかもしれません．これは，PCIデバイスにとって深刻です．たとえ，PCIデバイスがアクセスしようとしているページアドレスを知ったとしても，ページアウトされてしまい，ほかのプロセスの情報がその場所に書かれてしまっているかもしれません．このような状況を避けるために，ページをピンダウン(固定)する必要があります．

(3) CPUに対象となるページがキャッシュされてしまうと，正しく値の更新が行われない可能性がある

PCIデバイスがあるページを読み書きする場合，CPUがそのページをキャッシュしてしまっているときの状況を考えてみま

Column 4

ゼロコピー通信とは？

通信を主とするPCIデバイスでは，最近，ユーザー空間に用意されたメッセージをそのデバイスが直接アクセスしてネットワークに送信する，または，受信する手法がはやっています．

つまり，メッセージはTCP/IPなどの処理を介することなく，ネットワークに放出されていきます．メッセージの受信時には，PCIデバイスが直接，ホストメモリをアクセスし，プロトコル層をバイパスします．このようにすることで，プロトコル層でのコピー操作を回避し，デバイス本来の性能を引き出すアクセスを実現しています．

このような，まったくコピー操作をともなわない通信のことをゼロコピー通信と呼びます．

ゼロコピー通信はピンダウン機能を元に実装されており，インテリジェントなI/Oを実現できるPCIデバイスがそのマスタアクセスによる転送に関与します．

クラスタ計算機環境などの計算ノードが地理的に狭いところでネットワークで接続された環境で，その効果を発揮しています．

しょう．PCIデバイスから値を書き換えるとき，プロセスは値の更新を知ることができない場合もあり，さらに，CPUによるキャッシュラインフラッシュにより，PCIデバイスによって書き換えられた値が書きつぶされてしまうかもしれません．反対に，PCIデバイスがプロセスのメモリを読む場合では，最新のデータはキャッシュの中にあり，PCIデバイスが本来のデータを読めない場合があります．

このような状況に対応できるよう，PCIデバイスがアクセスする対象となるページは，キャッシュ不可にしなくてはなりません．

（4）領域は離散的な複数の物理ページをまたぐことがある

mallocなど一般的な領域確保の関数でユーザー空間から確保された領域は，ユーザー仮想アドレスおよびカーネル仮想アドレスでは連続アドレスとなりますが，物理アドレス空間では離散的になります．これは，メモリマネージャが物理ページを離散的に割り当てるためです．PCIデバイスが自律的にアクセスする場合は，この離散的なアドレス空間を把握してアクセスできる機能が必要になります．または，ユーザーアプリケーションがDMAコントローラを制御する場合は，ページサイズごとに途切れた転送をすることが必要になります．

メモリページをピンダウンするということは，以上の四つの問題点を解決し，PCIデバイスが安定してアクセスできる状況を作り出すことにあります．

●ピンダウンの利点/欠点は？

ではピンダウンすると，どのような利点があるのでしょうか？

PCIデバイスとプロセスの間でデータを共有する場合，一般的にはデバイスドライバ内でデバイスがアクセスするデータ領域を用意し，DeviceIoControl関数でそのデータを読み書きするのですが，この方法ではカーネル内のメモリは容量が制限されているため大きなデータをやり取りできず，カーネル内でデータコピーが発生するという制限があります．

プロセスのメモリをピンダウンし，PCIデバイスから直接アクセスを許すと，これらの問題を回避することができます．つまり，PCIデバイスはプロセスのアクセスするメモリアドレスへ直接アクセスでき，OSの厚いレイヤを飛び越し，コピーなしにデータを共有することが可能になります．

しかし，ピンダウンにも欠点があります．ピンダウンするには，ページをロックするためのメモリマネジャの時間を無視することはできません．PCIデバイスからアクセスするたびに，プロセスはピンダウンをカーネルに要求すると，そのオーバヘッドで実行速度が落ちることになります．

また，ピンダウンすることにより，ページングされないメモリ領域が物理メモリに存在することになることになります．すなわち，ピンダウンされた領域が多く存在するようになると，ほかのプロセスへ迷惑をかけることになります．OSが安定して動作できるよう，適当なタイミングでピンダウンを解放することが必要になります．

> **Column 5**
>
> **Windows2000/XPでのページサイズは4Kバイト固定**
>
> i386以降のCPUを搭載したマシンで動作するWindowsのページサイズはいくつでしょうか？　バージョンによって異なるのでしょうか？　そんな疑問がある人もいると思います．
>
> Linuxでは，getpagesize()システムコールでユーザープロセスがページサイズを取得する方法がありますが，Windowsではありません．DDKのntddk.hを見てみると，次のようなページサイズの定義があります．
>
> ```
> #define PAGE_SIZE 0x1000
> ```
>
> つまり，4Kバイトで固定で定義されているのです．

2.2　Windowsでのメモリマネージメント

Windowsでピンダウン機能を実現する場合に避けて通れないアドレス管理について説明します．Windowsには三つのアドレスの種類があります．それぞれの関係を次に示します．

（1）ユーザー仮想アドレス

ユーザーアプリケーションが扱うアドレス空間です．Windowsのアプリケーションは，すべてこのアドレス空間で動作します．DeviceIoControl関数などで，カーネル空間にポインタを渡すと，そのポインタの指す領域はカーネル仮想アドレス空間にマップされます．

（2）カーネル仮想アドレス

Windowsカーネル空間で管理されるアドレス空間です．カーネルモードデバイスドライバを含むカーネルと協調し動作するプログラムは，このアドレス空間を扱います．このアドレス空間はユーザー仮想アドレス空間と分離されているので，ユーザーモードアプリケーションから渡されたポインタなどはすべて，カーネル仮想アドレスに変換されます．このとき，サブシステムがアドレス変換を行っています．

（3）物理アドレス

CPUバスが扱うアドレスです．CPUの周辺デバイスはこのアドレス空間でアクセスします．このアドレスをPCIデバイスはプロセスから受け取り，I/Oアクセスを行うことができます．物理アドレスはカーネル仮想アドレスから変換することができますが，ユーザー仮想アドレスから直接変換することができません．

2.3　どのようにピンダウンするか？

ユーザーアプリケーションのメモリに関しては，ユーザー仮想アドレスで管理されています．ここでの操作は，まずユーザー仮想アドレスから物理ページアドレスを求めるということ

です．しかし，このアドレス変換はカーネル空間，つまり，デバイスドライバを介さないとできません．そこで，DeviceIo Contorol関数に注目します．

● ピンダウンの方法

デバイスドライバの機能として，DeviceIoControl関数のインターフェースを実現するには，そのディスパッチルーチンを記述することとともに，そのインターフェースを定義することが必要です．DeviceIoControl関数の引き数を見てみると，I/Oコントロールコード，入力バッファのポインタとその領域のサイズ，入力バッファのポインタとその領域のサイズです．

I/Oコントロールコードは，DDKで提供されるCTL_CODEマクロを用いて指定されます．じつは，CTL_CODEの指定の仕方いかんで，出力バッファをピンダウンしてくれる機能があります．CTL_CODEマクロで指定できる入出力バッファのアクセス方法には3種類あります．

（1）METHOD_BUFFERED

I/Oバッファの領域を，一度IRPの一時バッファに格納し，IRPの完了で出力バッファに書き戻す方法です．この方法では，PCIデバイスからの直接アクセスはできません．

（2）METHOD_IN_DIRECT，METHOD_OUT_DIRECT

出力バッファをピンダウンし，ページアドレスのリストを作成します．キャッシュ禁止になり，外部デバイスからのI/Oを待ちかまえる形になります．この方法の場合，PCIデバイスが直接アクセスすることが可能です．

（3）METHOD_NEITHER

引き数としてとられたI/Oバッファは無視されます．I/Oコントロールコードのみがドライバに渡ります．

この中の（2）を指定すると，サブシステムは引き数のバッファをピンダウンし，ページアドレスリストを作成してくれます．このピンダウン状態はIRPが完了するまで有効です．

ここで，DeviceIoControl関数は非同期で実行される必要があります．同期的に実行されると，DeviceIoControl関数はIRPの完了を待ち続けてしまうので，アプリケーションがデッドロックしてしまうことに加え，IRPが完了するとピンダウンされたページは，ピンダウン状態から解放されてしまいます．そこで，IRPを完了せずに，DeviceIoControl関数を終了することが必要です．

これには，CreateFile関数で，ファイルハンドルに関するファイル操作は非同期で行うよう，属性としてFILE_FLAG_OVERLAPPEDを指定することで，IRPをペンディングのまま終わらせ，出力バッファに与えたメモリ領域をピンダウンしたままにできます．

● 物理ページアドレスの取得

DeviceIoControl関数でメモリをピンダウンすることはできることがわかりましたが，ユーザーアプリケーションでそのメモリの物理ページアドレスのリストを取得できません．そこでもう一つ，物理ページアドレスリストを取得するI/Oコントロールコードを定義する必要があります．

この処理に関しては，ユーザーアプリケーションがテーブル取得のためのバッファポインタとピンダウンされているユーザー仮想アドレスをDeviceIoControl関数に渡し，ドライバ内でページアドレスを返します．したがって，ピンダウンされた領域を管理するデータ構造がドライバ内に必要になります．

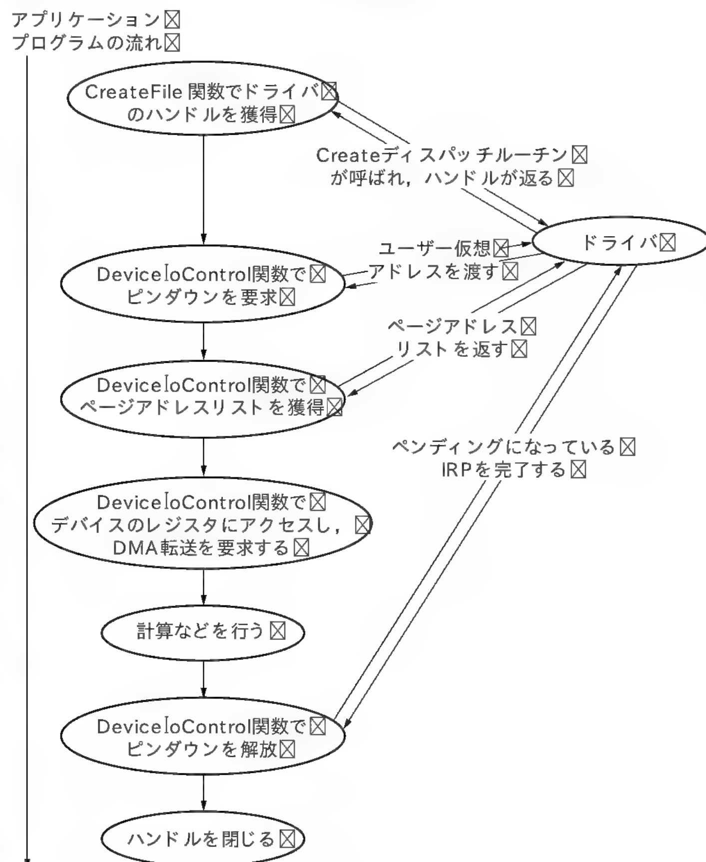
● ピンダウンの解放

ピンダウンされた領域を開放するには，ペンディング状態になっているIRPを解放すれば，ページング可能状態に戻され，さらに，キャッシュ可能な領域に戻ります．

● アプリケーション作成ガイドライン

ユーザーアプリケーションからどのようにバッファ管理をすればよいかを考えます．ドライバのハンドルを取るところから，ピンダウンされた領域の解放までの一連の流れについて，図4にまとめます．

〔図4〕ユーザーアプリケーションの処理の流れ



3 PCIDMACドライバの実装

それではいよいよ，PCIベースのDMAコントローラを制御するためのPCIDMACドライバの実装について説明していきます．このドライバには，

1) インストール用INFファイル
2) クラスインストーラDLL
3) ドライバSYSファイル

が必要になります．それぞれのファイルの実装について順に説明していきます．

3.1 INFファイル

INFファイルは構造化された文法をもっているのは確かですが，読みなれるまで，たいへんな思いをします．‘＝’の左辺と右辺の扱いが変わったり，必ずしも式でなかったりします．このような書き方なんだなと思い，暗記するつもりで解析してください．

このINFファイルでは，ドライバの情報をウィザードに表示，クラスのインストールの指示，サービスのインストールの指示の三つのことを指定します．

●[Version]セクション

ここには，サポートするOS系列，デバイスクラス，ドライバを提供するベンダ情報がかかれています．これらの情報はドライバインストール時にウィザードに表示される情報を提供します．クラスはPCIDMACクラスでそのGUIDを独自生成し，指定しています．

カタログファイルはマイクロソフトのドライバテストに合格しないともらえません．またカタログファイル名を指定しないとエラーになります．ここでは適当に名前を付けています．`Signature`エントリではNT系OSを指定しています．

●[Manufacturer]セクションと[PCIVENDER]セクション

ここに，このドライバがサポートするデバイスのリストを書きます．ウィザードはここに書かれた情報を元に，デバイス一覧を表示します．

●[ClassInstall32]セクション

クラスインストーラのインストールを促すセクションです．`AddReg`エントリでレジストリに書く情報を指定し，`CopyFiles`でDLLを指定します．

●[PCIDMAC.NT.Services]セクション

ドライバをシステムサービスとして登録し，起動時に自動的にロードされるようにする指示です．このところの指定は複雑なので，DDKドキュメントで確認してから指定してください．

●[Strings]セクション

ここには文字列定義がされています．ウィザードに出てくる文字列がほとんどです．

3.2 クラスインストーラ

クラスインストーラは，じつは何もしません．しかし，クラスアイコンを設定するためにこのDLLが必要になります．PnPマネージャが`classinst.c`で示す`InstallFunction`を，DLLの関数である`PCIDMAControllerClassInstaller`に渡してきます．この値にしたがい適宜処理をします．クラスインストーラに限ってはユーザーモードで実行されます．

アイコンはリソースファイルにファイル名を指定し，DLLに埋め込みます．デバイスマネージャを開いたときにはこのDLLを読み，クラスアイコンを表示しています．

> **Column 6**
>
> **INFファイルの文法チェック方法**
>
> INFファイルを書いていると，ふと気になるのは，「本当に文法があっているのか？」，「セクション間の依存関係に穴がないか？」ということでしょう．
>
> DDKにはこれらの疑問を解決するため，infchkというINFファイル用文法チェッカが用意されています．このチェッカを起動するには，Perlが必要になるので，ダウンロードして使ってください．
>
> クラス名がシステム標準でない場合は，GUIDが一致しないのでエラーとして通知されますが，問題ありません．これは，システム標準クラスのみをサポートしているためです．
>
> ただし，このツールでチェックできるのはあくまでINFファイルの文法なので，その中の記述が論理的に合っているかどうかは，あくまで人間が確認しなければなりません．

3.3 ドライバ

ドライバを作成する際には，`DeviceIoControl`関数からコントロール関数を参照できるように，ドライバ本体(`.c`)とヘッダファイル(`.h`)を分けてください．`.h`ファイルはユーザーアプリケーションで参照されます．PCIDMACでは`DMA_drv.c`と`DMAC_ioctl.h`から構成されます．

● DMAC_ioctl.hヘッダファイル

ヘッダファイル中では，ドライバ内の情報をユーザーアプリケーションからも共有できるように構造体の宣言が書かれています．ドライバに対し，`DeviceIoControl`関数で，`PCIDMAC_GET_DEV`コントロールを指定すると，この`struct PCIDMAC_Dev`が返ります．`#ifdef _NTDDK_`は，このドライバ中での情報がユーザーアプリケーションでも読めるように異なる定義をしています．

また，ここでは`DeviceIoControl`関数から指示できるコントロールコードを定義しています．`PCIDMAC_PINDOWN_MEMORY`の定義では，`PCIDMAC_DIRECTOUT_CTL_CODE`マクロを用い，サブシステムによって，DevicoloControlの引き数であるバッファをロックしてもらいます．このマクロでは，後述するMDLがサブシステムにより作成されます．`PCIDMAC_CTL_CODE`マクロで定義したコントロールコードに関しては，`DeviceIoControl`関数の引き数である入出力バッ

ファを一度バッファしてドライバへと渡します．

● DMAC_drv.cドライバ本体(リスト3)

1）DriverEntryルーチン，PCIDMACAddDeviceルーチン

PnPマネージャはドライバのエントリポイントのDriverEntryルーチンにジャンプし，設定が始まります．DriverEntryでは，PnPマネージャが用意したDriverObjectにディスパッチルーチンを登録することだけがおもな仕事です．

DriverEntryのあと，PCIDMACAddDeviceルーチンが呼ばれます．PCIDMACAddDeviceルーチンでは，ドライバが用いるリソースの設定をしています．IoCreateDevice関数でドライバのシステム内でのインスタンス領域を作ります．DeviceIoControl関数でユーザー空間からアクセスできるように，IoRegisterDeviceInterface関数でデバイスのインターフェースを登録します．これにより，GUIDでこのデバイスへのインターフェースを獲得できるようになります．そして，デバイス独自の領域(DeviceExtension)を初期化します．DeviceExtensionはドライバが独自に用いる自由領域になっています．

この領域へのポインタはDriverObjectの一部に登録されるため，どのディスパッチルーチンからも参照できます．さらに，デバイススタックに下位のデバイススタックを指定し，デバイスのインターフェースを指定します．最後に，

```
DeviceObject->Flags &=
                ~DO_DEVICE_INITIALIZING;
```

で，ドライバの初期化が完了したことをマークします．

2）PCIDMACPnpルーチン

初期化が完了すると，PCIDMACPnpルーチンが呼ばれます．このとき，PnPマネージャからリソースが渡ってくることが重要な点です．ルーチンの引き数のIRPに含まれるMinorFunctionがIRP_MN_START_DEVICEの場合，デバイスの動作を開始する要求で，デバイスのリソースが渡ってきます．その種類にしたがって，リソースの登録を行っていきます．

メモリマップドリソースの場合，MmMapIoSpace関数で領域をカーネルアドレスにマップします．I/Oマップドリソースの場合はマップせずにREAD_PORT_BUFFER_*()マクロなどを用いて直接読み書きするため，ポート番号だけ記録しておきます．

ここで渡ってくるリソースの順番ですが，PCIコンフィグレーション空間のベースアドレスレジスタ0から5に順に渡ってくるようです．よってインデックスをメモリマップド領域とI/Oマップド領域のそれぞれにふっておきます．

IRQに関しては，このドライバでは用いていないので，記録だけにとどまります．

3）PCIDMACDeviceControlルーチン

ユーザーアプリケーションからDeviceIoControl関数が呼ばれると，このルーチンが実行されます．IoGetCurrentIrpStackLocation関数でデバイススタックを獲得し，その中にあるコントロールコードを調べ，各要求にしたがった処理を実行していきます．

このとき，PCIDMAC_PINDOWN_MEMORY以外の要求に関しては，I/Oバッファはバッファされ，コピーされてきます．このバッファの大きさは，入力と出力の大きいほうにあわせた領域が確保されています．入力も出力も同一のバッファに読み書きするわけですが，書き込まれた値はDeviceIoControl関数の出力バッファへとコピーされて，ユーザー空間に渡ります．出力データを書く際には，入力データを書き換えてしまいますが，バッファされているため，元のユーザー空間のデータは保護されています．

一方，PCIDMAC_PINDOWN_MEMORYの場合は，ユーザーアプリケーションが与えたバッファがそのままロックされ，ドライバに直接渡ってきます．この渡ってくるバッファはDeviceIoControl関数に与えられた出力バッファとなります．さらに，PCIDMACDeviceControlルーチンに渡されたバッファには，MDLと呼ばれるバッファの情報が格納された構造体が付随してきます．MDLにはページ番号と呼ばれるものが付いています．ページ番号とは，物理メモリの0番地から順に振ったページアドレスをページサイズで割ったものです．この番号にページサイズをかけることで，ページアドレスが求められます．

PCIDMAC_PINDOWN_MEMORYが要求された際には，IRPがユーザー仮想アドレスをキーとした線形リストに保存され，未処理状態であることを示すSTATUS_PENDINGが返されます．このIRPはPCIDMAC_RELEASE_PINDOWN_MEMORY要求か，ユーザー空間で獲得されたハンドルが閉じられるときのPCIDMACCleanupルーチンが呼ばれたときに解放処理を行います．各要求を処理したあとは，PCIDMAC_PINDOWN_MEMORY以外の要求に関しては，IRPを完了させます．

たとえ，DeviceIoContorol関数が非同期で呼ばれても，このルーチン内でデータ取得が行われてリターンしている場合には，同期として呼ばれた場合と同じになります．

4）PCIDMACCreate，PCIDMACCleanup，PCIDMACClose，PCIDMACUnloadルーチン

PCIDMACCreate，PCIDMACCloseに関してはとくにやることがないので，IRPを完了させて返るだけですが，PCIDMACCleanupルーチンには，現在，これらのルーチンの実行対象となってるプロセスに関して，ロックされた領域を開放する役目があります．

不正なアクセスや，CloseHandle関数や，クローズしないままでのプロセスの終了の際にはPCIDMACCleanupルーチンが呼ばれ，ロックされた領域を開放します．

5）PCIDMACPowerルーチン

電源管理のルーチンですが，このドライバでは電源管理をしていないので，IRPを完了させて返ります．この完了処理に関しては，通常のIRPと扱う関数が異なるので注意が必要です．

〔リスト3〕作成したドライバ(DMAC_drv.c)

```
// 使用許諾
// 1.このソースコードならびに，このソースコードから生成されるオブジェクト
//   コードを用いる際にはこの許諾を得た上で使用しているものとします.
// 2.このコードの著作権は山際に帰属しますが，コードの利用・配布はフリーです.
//   配布の際は必ず本コードを用いていることを明記してください.
// 3.コンパイルおよび動作上の不具合，および，利用者の環境における損害に
//   関しては，著作者はその一切の責任を負いません.
// 4.著作者は本ファイルに含まれるコードの保守・改良などの責任は一切負わない
//   ものとします.
// 5.本コードは著作者の有する実験環境のみでテストされており，ほかのすべての
//   環境下で動作する保障は一切ありません.
  ～中略～
NTSTATUS
DriverEntry(
        IN PDRIVER_OBJECT DriverObject,
        IN PUNICODE_STRING RegistryPath
        )
{
  ～中略～
  // ディスパッチルーチンを初期化します.
  DriverObject->MajorFunction[IRP_MJ_CREATE]         = PCIDMACCreate;
  DriverObject->MajorFunction[IRP_MJ_CLOSE]          = PCIDMACClose;
  DriverObject->MajorFunction[IRP_MJ_CLEANUP]        = PCIDMACCleanup;
  DriverObject->MajorFunction[IRP_MJ_PNP]            = PCIDMACPnp;
  DriverObject->MajorFunction[IRP_MJ_POWER]          = PCIDMACPower;
  DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]
                                                  = PCIDMACDeviceControl;
  DriverObject->DriverExtension->AddDevice           = PCIDMACAddDevice;

  return Status;
}

  ～中略～

NTSTATUS
PCIDMACDeviceControl(
              IN PDEVICE_OBJECT DeviceObject,
              IN PIRP Irp
              ){
  ～中略～
  // IO_CTLコードに従い，それぞれの処理を実行する
  switch(code){
  case PCIDMAC_PINDOWN_MEMORY: // メモリをピンダウンします
    if(Irp->MdlAddress == NULL || inputBufferLength < sizeof(ULONG)){
     //入出力バッファに問題があるとき
      ntStatus = STATUS_INVALID_BUFFER_SIZE;
    }
    else{
      ～中略～
      ntStatus = _add_to_pindown_list(Irp->MdlAddress,
                          *((ULONG *)ioBuffer),
                          &list_tmp);
      if(ntStatus != STATUS_SUCCESS){
      }
      else{
    // ここではアドレスは取れません
    // この場合は要求を終わりにしないでロックしっぱなしにする
    info = 0;
    Irp->IoStatus.Status = STATUS_SUCCESS;
    Irp->IoStatus.Information = info;
    list_tmp->Irp = Irp;
    goto return_without_complete;
      }
    }
    break;
    ～中略～
  case PCIDMAC_RELEASE_PINDOWN_MEMORY:
    ～中略～
      ntStatus = _find_pindown_list(*((ULONG *)ioBuffer), &list_tmp);

      if(ntStatus == STATUS_SUCCESS){
    _delete_pindown_list_by_uva(*((ULONG *)ioBuffer));

    // ここで，要求を完了させてロック状態から開放させる
    IoCompleteRequest(list_tmp->Irp, IO_NO_INCREMENT);
    info = 0;
      }
    }
    break;
  case PCIDMAC_SET_DMA_REQ: // DMA要求をセットします
    ～中略～
      _internal_set_dma((PCIDMAC_dma_req *)ioBuffer, deviceExtension);

      _sync_dma(0,deviceExtension);

      info = sizeof(PCIDMAC_dma_req);
      ntStatus = STATUS_SUCCESS;
    }
    break;
  default:
    ntStatus = STATUS_INVALID_DEVICE_REQUEST;
  }
  ～中略～
  return ntStatus;
}

NTSTATUS
PCIDMACAddDevice(
           IN PDRIVER_OBJECT   DriverObject,
           IN PDEVICE_OBJECT   PhysicalDeviceObject
           ){
  ～中略～
  // デバイスオブジェクトを作成します
  ntStatus = IoCreateDevice( DriverObject,
                    sizeof(DEVICE_EXTENSION),
                    NULL,
                    FILE_DEVICE_UNKNOWN,
                    0,
                    FALSE,
                    &DeviceObject);
  if (ntStatus != STATUS_SUCCESS) {
    return ntStatus;
  }
  // インターフェースクラスを作成
  ntStatus = IoRegisterDeviceInterface(PhysicalDeviceObject,
                        &GUID_DEVINTERFACE_PCIDMAC,
                        NULL,
                        &SL_name_unicode);
  if (ntStatus != STATUS_SUCCESS) {
    return ntStatus;
  }
  // デバイスエクステンションを初期化
  deviceExtension = DeviceObject->DeviceExtension;
  RtlZeroMemory(deviceExtension, sizeof(DEVICE_EXTENSION));
  // DeviceObjectをバックアップ.
  deviceExtension->DeviceObject = DeviceObject;
  // デバイスをデバイススタックに追加
  deviceExtension->StackDeviceObject =
    IoAttachDeviceToDeviceStack(DeviceObject, PhysicalDeviceObject);
  ～中略～
  DeviceObject->Flags &= ~DO_DEVICE_INITIALIZING;
  return ntStatus;
}

NTSTATUS
PCIDMACPnp(
        IN PDEVICE_OBJECT   DeviceObject,
        IN PIRP             Irp
        ){
  ～中略～
  switch (IrpSp->MinorFunction) {
  case IRP_MN_START_DEVICE:
    //このリクエストが送られてきたときに，PCIバスの情報が送られてきます
    AssignedResources = IrpSp->Parameters.StartDevice.AllocatedResources;
    AssignedResourcesTranslated =
          IrpSp->Parameters.StartDevice.AllocatedResourcesTranslated;
    deviceExtension->AssignedResources = AssignedResources;
    deviceExtension->AllocatedResourcesTranslated =
                                        AssignedResourcesTranslated;
    ～中略～
    if(AssignedResourcesTranslated->Count > 0){
      // リソースがわたってきます.
      for (i=0;i < AssignedResourcesTranslated->
                          List[0].PartialResourceList.Count;i++ ){
    switch (AssignedResourcesTranslated->
              List[0].PartialResourceList.PartialDescriptors[i].Type){
      ～中略～
    case CmResourceTypeMemory:          //リソースがメモリだったとき
      deviceExtension->mapped_mem_size[deviceExtension->maxmemid] =
       AssignedResourcesTranslated->
       List[0].PartialResourceList.PartialDescriptors[i].u.
                                    Memory.Length;
      deviceExtension->mapped_mem_base_phy[deviceExtension->maxmemid] =
        ((AssignedResourcesTranslated)->
        List[0].PartialResourceList.PartialDescriptors[i].u.
                                    Memory.Start.u.LowPart);
      deviceExtension->mapped_mem_base_kv[deviceExtension->maxmemid] =
      MmMapIoSpace((AssignedResourcesTranslated)->
      List[0].PartialResourceList.PartialDescriptors[i].u.Memory.Start,
      AssignedResourcesTranslated->
      List[0].PartialResourceList.PartialDescriptors[i].u.Memory.Length,
                                    MmNonCached);
      deviceExtension->maxmemid ++;
      break;
    case CmResourceTypePort:            // リソースがポートだったとき
      deviceExtension->mapped_port_size[deviceExtension->maxportid] =
       AssignedResourcesTranslated->
       List[0].PartialResourceList.PartialDescriptors[i].u.Port.Length;
      deviceExtension->mapped_port_base[deviceExtension->maxportid] =
        ((AssignedResourcesTranslated)->
        List[0].PartialResourceList.PartialDescriptors[i].u.Port.Start.
                                    u.LowPart);
      deviceExtension->maxportid ++;
      break;
    ～中略～
}
```

3.4 インストール

インストールする前に，PCIデバイスがWindowsがデフォルトでサポートするデバイスクラスとして設定されていないことを確認してください．たとえば，PCIのデバイスファンクションに標準RAMクラスが設定されていると，Windowsは自動的にシステム内にあるドライバを割り当ててしまうため，新たなクラスを定義することができなくなります．

デバイスを挿入した後，Windowsが起動すると，デバイスマネージャを開きます．「その他のデバイス」に妥当な名前が付けられデバイスが認識されていると思います．このデバイスが先ほどPCIスロットに接続したデバイスです．このデバイスにインストールしていきます．

デバイスを右クリックし，プロパティウィンドウを開きます．このウィンドウ中のドライバの更新ボタンをクリックし，ドライバのインストールを始めます．

この際に，ドライバの自動認識を使用しないで，手動でデバイスクラスとINFファイルを選んでください．デバイスクラスを指定する際には，「その他のデバイス」デバイスクラスでも選んでおけば大丈夫です．INFファイルの位置を選ぶときには，「ディスク使用」ボタンをクリックし，INFファイルの位置を指定してください．

先に進んでいくと，まず，クラスインストーラDLLが催促されるので，ファイルを指定してください（INFファイルと同一フォルダにDLLがある場合には催促されない）．次に，ドライバSYSファイルが催促されます．

以上の手順が完了すると，デバイスマネージャにデバイスクラスとデバイス名が表示されます．デバイスのプロパティを見ると，PCIデバイスのリソースが割り当てられていることが確認できるでしょう．

3.5 アンインストール

● ドライバの完全削除

ドライバの削除は，まず，デバイスマネージャでPCIDMACクラスのドライバを削除してください．さらに，

```
C:¥WINNT¥system32¥drivers¥DMA_drv.sys
```

を削除します．INFファイルは，

```
C:¥WINNT¥inf
```

にある`oem??.inf`に名前が変えられて保存されています．??はWindowsが事前にもつドライバ以外のものをインストールした順番で番号が振られていきます．中身を見て，間違えないように削除してください．また同一の名前の`oem??.PNF`も削除してください．

● クラスの完全削除

クラスの削除に関しては，DLLとレジストリ項目の削除を行います．まずレジストリエディタを開きます．次のようなPCIDMACクラスがあるので，それを削除します．

```
¥¥HKEY_LOCAL_MACHINE¥SYSTEM¥
        CurrentControlSet¥Ccontrol¥Class¥
        {BFE02D55-40D5-42c6-9A80-2907BEC5AE12}
```

DLLは，

```
システムドライブ:¥WINNT¥system32
```

に保存されています．この中にあるDLL（`PCIDMACcls.dll`）を，ドライバの削除後に削除します．

4 DMAコントローラを制御してみよう

～DMAコントローラの制御方法とPCIDMACライブラリ～

今月号付属CD-ROM InterGiga No.32には，PCIDMACライブラリが収録されています．このライブラリは前述した複雑な手順を隠ぺいするようにしてあります．アプリケーション作成者はドライバの詳細を知ることなく，プログラムが可能になっています．それぞれのインターフェース関数を表2に示します．

PCIDMACライブラリでは，

（1）DMA設定とポーリングによる同期をユーザーアプリケーションから行った場合

（2）DMA設定をドライバ内部で行いポーリングによる同期をユーザーアプリケーションから行った場合

（3）DMA設定とポーリングによる同期をドライバ内から行った場合

の3種類の方法を可能にしています．

それぞれのモードに変更するには，`PCIDMA.c`（リスト4，p.92）の中のⒶとⒷ部分のコメントを外してビルドを再度実行してください．

Column 7

チェーンドまたはスキャッタギャザDMAとは？

ホスト主導でDMA転送をページごとにかけるコストは，`DeviceIoControl`関数を連発することになるので大きなものになります．このような，離散的な場所に連続データを転送する手段としてチェーンドDMA，またはスキャッタギャザDMAがPCIデバイスに実装されていることがあります．チェーンドDMAとは，複数回実行するDMAをディスクリプタと呼ばれる構造体に設定し，ディスクリプタ間を物理アドレスのポインタでつなぎ合わたリストを作成し，1度のDMAスタート要求で，そのリストを一気に実行する方法です．スキャッタギャザDMAとは，仮想アドレスの連続している領域の物理ページアドレスの配列を作成し，一度のスタート要求で，DMAが一気に開始される方式です．

Intel EtherExpressProやSMC EPIC100などのEthernet PCIボードでは，チェーンドDMAがデバイスによりサポートされ，離散領域へのDMA転送を援助しています．

以上3種類の場合について，アプリケーションプログラムを書く上で関数の呼ばれる手順を図5に示します．(3)の場合，ユーザーアプリケーションがDMA完了確認を再度行うと無限ループにはまる場合があるので(たとえば，完了確認した際にDMA完了フラグがリセットされてしまうような場合)，本ドライバを移植した場合には注意してください．

付属CD-ROMのサンプルプログラム(PCIDMAC_sample.c)にページごとにDMAをかける方法が記述されているので，参考にしてください．このサンプルでは，領域の先頭に関してはオフセットがあるかもしれないので，オフセット計算をしています．ユーザー仮想アドレスの下位12ビットは対応する物理ページオフセットとして扱えるので，それを抽出して，先頭ページのオフセットしています．

5 性能評価実験

PCIDMACライブラリを使い，

(1) 実験1: DMA設定とポーリングによる同期をユーザーアプリケーションから行った場合
(2) 実験2: DMA設定をドライバ内部で行いポーリングによる同期をユーザーアプリケーションから行った場合
(3) 実験3: DMA設定とポーリングによる同期をドライバ内から行った場合

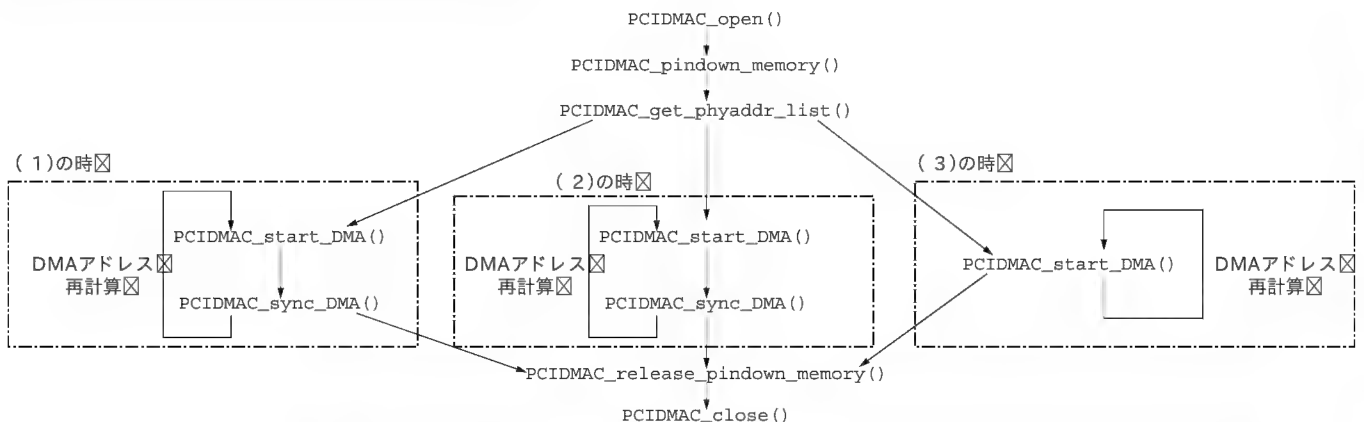
の3種類の実験をしました．それぞれの場合を表3に示します．結果は16Mバイトのデータを20回繰り返し転送したときの1回あたりの平均値です．

(1)の場合，DMA設定のすべてがDeviceIoControl関数で行われるため，レジスタアクセスごとにその関数が呼ばれます．(2)の場合，DMA設定に関してはドライバ内部でマップアドレスに連続してアクセスされます．ポーリングに関しては，DeviceIoControl関数を介して行われます．(3)の場合はDMAの設定と完了待ちポーリングをドライバの中で行います．このときには，DeviceIoControl関数はDMA起動ごと(つ

〔表2〕作成したドライバとユーザーアプリケーションとのインターフェース関数一覧

int PCIDMAC_open()　ライブラリをオープンする	
引き数	なし
戻り値	0以外が返るとエラー
備　考	アプリケーションの最初で一度だけ必ず呼ぶ
void PCIDMAC_close()　ライブラリをクローズする	
引き数	PCIDMAC_handle *handle ……オープンで獲得したハンドル
戻り値	なし
PCIDMAC_phyaddr_list *PCIDMAC_pindown_memory()　与えたメモリをピンダウンする	
引き数	void *buf_base_addr …… ユーザー空間のバッファポインタ unsigned long size …… サイズ
戻り値	物理ページアドレスリストを返す．NULLが返るとエラー
int PCIDMAC_release_pindown_memory()　ピンダウン済みのメモリを開放する	
引き数	PCIDMAC_pindown_memoryに同じ
戻り値	成功すると1，失敗すると0以外が返る
int PCIDMAC_start_DMA()　DMAを開始する	
引き数	int dmac_id …… DMAコントローラへのインデックス unsigned long start_phy_addr … スタートアドレス．物理アドレスを指定 unsigned length …… DMA転送長
戻り値	0でない値が返るとエラー
void PCIDMAC_sync_DMA()　DMAと同期を取る	
引き数	int dmac_id …… DMAコントローラへのインデックス
戻り値	0でない値が返るとエラー
int PCIDMAC_check_DMA()　DMAコントローラが転送中であるかチェックする	
引き数	int dmac_id …… DMAコントローラへのインデックス
戻り値	0でない値が返ると転送中
PCIDMAC_phyaddr_list *PCIDMAC_get_phyaddr_list()　ピンダウンメモリの物理アドレスリストを獲得する	
引き数	void *buf_base_addr …… バッファへの仮想アドレス int size …… バッファのサイズ
戻り値	アドレスリストへのポインタを返す．NULLが返るとエラー
備　考	アドレスリストにはページアドレスが入っているので，サイズ計算のときには必ずオフセットを計算する
int PCIDMAC_verify_pci_mem()　PCIデバイス上のデータとデータベリファイする	
引き数	unsigned char *buf …… ホストメモリのデータをさすポインタ unsigned long pci_addr …… PCIデバイス上のアドレスオフセット unsigned long size …… サイズ
戻り値	0でない値が返るとエラー

〔図5〕アプリケーション作成時の関数の呼ばれる手順



Prologue Chapter 01 Chapter 02 Chapter 03 Appendix 01 Chapter 04 Chapter 05 Appendix 02 Chapter 06 Appendix 03

〔リスト 4〕ドライバ DMA_drv を呼び出すサンプルユーザーアプリケーション(PCIDMA.c)

```
// PCI DMAコントローラライブラリ(無償配布版)
// (c)Shinichi Yamagiwa(yama@pdmfc.com)
// 使用許諾
// 1.このソースコードならびに,このソースコードから生成されるオブジェクト
// コードを用いる際にはこの許諾を得た上で使用しているものとします.
// 2.このコードの著作権は山際に帰属しますがコードの利用/配布はフリーです.
// 配布の際は必ず本コードを用いていることを明記してください.
// 3.コンパイルおよび動作上の不具合,および,利用者の環境における損害
// に関しては,著作者はその一切の責任を負いません.
// 4.著作者は本ファイルに含まれるコードの保守・改良などの責任は
// 一切負わないものとします.
// 5.本コードは著作者の有する実験環境のみでテストされており,
// ほかのすべての環境下で動作する保障は一切ありません.

～中略～
// ここのところ,モード切替
//#define DMA_IS_SET_IN_APP //この行を生かすと,ユーザ空間で DMA 設定    Ⓐ
#ifndef DMA_IS_SET_IN_APP
#define DMA_IS_SET_IN_DRIVER
#endif
//#define DMA_SYNC_IN_APP //この行を生かすと,ユーザ空間で DMA 完了確認    Ⓑ
#ifndef DMA_SYNC_IN_APP
#define DMA_SYNC_IN_DRIVER
#endif

～中略～
int PCIDMAC_open(){
～中略～
  hdi = SetupDiGetClassDevs (
                  (LPGUID)&GUID_DEVINTERFACE_PCIDMAC,
                  NULL,
                  NULL,
                  (DIGCF_PRESENT | // 現在存在するものだけ
                   // インターフェースクラス要求する
                   DIGCF_DEVICEINTERFACE));
～中略～
    file = CreateFile ( deviceInterfaceDetailData->DevicePath,
                        // 読み書きモードで開く
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE, //
                        NULL, // lpSecurityAttributes は無視
                        OPEN_EXISTING, // デバイスを開くので EXISTING
                        FILE_FLAG_OVERLAPPED, // 属性は特になし
                        &__internal_handle->overlap_struct);
～中略～
    return 0;
}

void PCIDMAC_close(){
  CloseHandle(__internal_handle->handle);
  free(__internal_handle);
}

PCIDMAC_phyaddr_list *PCIDMAC_pindown_memory(
                        void *buf_base_addr, unsigned long size){
～中略～
  ntStatus = DeviceIoControl(
                  __internal_handle->handle,
                  PCIDMAC_PINDOWN_MEMORY,
                  &base_addr_long,
                  sizeof(unsigned long),
                  buf_base_addr,
                  size,
                  &bytes,
                  &__internal_handle->overlap_struct
                  );
～中略～
  // アドレスリストを返します
  if((page_addr_list=PCIDMAC_get_phyaddr_list(buf_base_addr,
                                         size))==NULL){
    printf("Err: Can not get page address list.¥n");
    if(PCIDMAC_release_pindown_memory(buf_base_addr, size) != 0){
      printf("Fatal Err: Can not release pindown!¥n");
    }
    return NULL;
  }

  return page_addr_list;
}

int PCIDMAC_release_pindown_memory(
                        void *buf_base_addr, unsigned long size){
  ～中略～

  ntStatus = DeviceIoControl(
                  __internal_handle->handle,
                  PCIDMAC_RELEASE_PINDOWN_MEMORY,
                  &base_addr_long,
                  sizeof(unsigned int),
                  buf_base_addr,
                  size,
                  &bytes,
                  &__internal_handle->overlap_struct
                  );

  if (!ntStatus) {
    printf("RELEASE: Ioctl failed with code %d¥n", GetLastError());
    return 1;
  }
  return 0;

}

int PCIDMAC_start_DMA(int dmac_id, unsigned long start_host_addr, unsigned
  long start_pci_addr, unsigned long length, int direction, int testmode){
    return _set_dma(dmac_id, start_host_addr, start_pci_addr,
                                      length, direction, testmode);

}

void PCIDMAC_sync_DMA(int dmac_id){
#ifdef DMA_IS_SET_IN_DRIVER
  while(_check_dma_state_clear(dmac_id) != 0) {}
#endif
}

void PCIDMAC_Reset_FIFO(int dmac_id){
      _write_control_reg(0x0C, 0x00FFC000);  // フラグクリア
}
int PCIDMAC_check_DMA(int dmac_id){
  if(_check_dma_state(dmac_id) != 0) return 1;
  return 0;
}

PCIDMAC_phyaddr_list *PCIDMAC_get_phyaddr_list(
                              void *buf_base_addr, unsigned long size){
  ～中略～
  // 最初ずれてるとき
  if(first_offset != 0){
    num_pages ++;
    remaining_size -=
     (((0x1000 - first_offset) > size) ? size : (0x1000 - first_offset));
  }
  // ページにかかる分を計算
  num_pages += (remaining_size / PAGE_SIZE);
  last_offset = remaining_size % PAGE_SIZE;
  remaining_size = last_offset;
  // 最後のオフセットを計算
  if(remaining_size != 0){
    num_pages ++;
    remaining_size -= last_offset;
  }
  ～中略～
  ntStatus = DeviceIoControl(
                  __internal_handle->handle,
                  PCIDMAC_GET_PINDOWN_PADDR_LIST,
                  pindown_pages,
                  sizeof(UINT)*num_pages,
                  pindown_pages,
                  sizeof(UINT)*num_pages,
                  &bytes,
                  &__internal_handle->overlap_struct
                  );
  ～中略～
  return page_addr_list;
```

まり 4K バイトごと)に1度呼ばれます.

(1)〜(3)にしたがって性能が上がるのがわかると思います.つまり,DeviceIoContorol 関数がオーバヘッドになっているのがわかります.性能を上げるには,できるだけ多くの DMA に関する処理をドライバ内で連続的に行わせるのがよいと考えられます.

しかし,ドライバ内に多くの処理をさせるとドライバの汎用性が低くなります.このトレードオフとなりますが,PCI で DMA コントローラを実装する場合には,性能を追求する場合がほとんどの場合だと思われるので,できるだけ処理をドライバに含める仕様にしたほうがよいでしょう.

〔表3〕転送レート比較結果

環境1(PentiumⅡ 266MHz/Intel 440LX/PC66 SDRAM 256Mバイト/Windows 2000)		
実験(1)	HOST→PCI	10.90Mバイト/秒
実験(1)	PCI→HOST	13.38Mバイト/秒
実験(2)	HOST→PCI	34.26Mバイト/秒
実験(2)	PCI→HOST	34.26Mバイト/秒
実験(3)	HOST→PCI	77.44Mバイト/秒
実験(3)	PCI→HOST	77.97Mバイト/秒
環境2(Athron 1.6GHz/AMD 754/DDR333 768Mバイト/Windows 2000)		
実験(1)	HOST→PCI	58.24Mバイト/秒
実験(1)	PCI→HOST	41.38Mバイト/秒
実験(2)	HOST→PCI	99.57Mバイト/秒
実験(2)	PCI→HOST	56.99Mバイト/秒
実験(3)	HOST→PCI	100.54Mバイト/秒
実験(3)	PCI→HOST	57.61Mバイト/秒
環境3(Celeron 1.7GHz(Socket 478)/ServerWorks GC-SL/DDR266 512Mバイト/Windows 2000)		
実験(1)	HOST→PCI	16.57Mバイト/秒
実験(1)	PCI→HOST	26.91Mバイト/秒
実験(2)	HOST→PCI	19.30Mバイト/秒
実験(2)	PCI→HOST	33.85Mバイト/秒
実験(3)	HOST→PCI	23.07Mバイト/秒
実験(3)	PCI→HOST	48.11Mバイト/秒

Column 8
NDISドライバの送受信処理

WindowsのNDISドライバを作成していると,「Windowsさん,そりゃないよ～」と思うことがあるかもしれません.送信するデータがNDISドライバに渡ってくるとき,IEEE802で定める1500バイト以下のデータが渡ってくるのですが,これを1回のDMAでは転送できないとわかるでしょう.なぜなら,その1500バイト以下のデータが複数の領域に分割されて渡されるからです.

考えられる効率的な方法としては,1度連続な領域にコピーし直し,整形してから転送するか,チェーンドまたはスキャッタギャザDMAを使うかです.

PCIデバイスが,チェーンドまたはスキャッタギャザDMAをサポートしていないと,前者の選択を余儀なくされます.Windowsをターゲットとするときには,チェーンドまたはスキャッタギャザDMAをデバイスに内蔵することが,勝利のカギ(?)となることでしょう.

6 ほかのPCIデバイスに移植するには?

ここで作成したドライバをほかのPCIデバイスに移植するには,次のような点に注意してください.

● INFファイルの変更

INFファイルについては次の点を変更してください.

1) クラスGUIDを新しいものに付け替える
2) インターフェースGUIDを新しいものに付け替える
3) `PCIVENDER`にボードメーカーの名前を入れる
4) `PCIDMACDesc`にデバイスの型や説明を入れる
5) `DevClassName`にクラス名を入れる
6) `PCIDMAC.SVCDESC`にクラス向けのサービスに表示するサービス名を入れる
7) `PCIDMACServiceDesc`にサービスの説明を入れる
8) `DiskId1`にドライバ用ディスクの名前を入れる
9) `[PCIVENDER]`セクションのPCIベンダIDとデバイスIDを変更する

● PCIDMAC関数の変更点

PCIDMACライブラリを他のPCIデバイスのDMAコントローラに移植するには,スタティック関数を移植することが必要になります.

`PCIDMAC.c`には,最下位関数がスタティック関数として定義されています.ファイルの最後にある`_set_dma`関数,`_check_dma_state`関数,`_check_dma_state_clear`関数,`_read_pci_mem`関数(デバッグ用),`_dump_regs()`(デバッグ用)を移植する必要があります.コード中に書かれたそれぞれのルーチンの説明に沿って,手順を書き換えてください.

● ドライバ内でDMA設定と完了確認

`DMA_drv.c`の`_internal_set_dma`関数にはDMAを設定する手順,`_sync_dma`関数にはDMAの完了を待つ手順を書いてください.

7 ドライバ使用上の注意

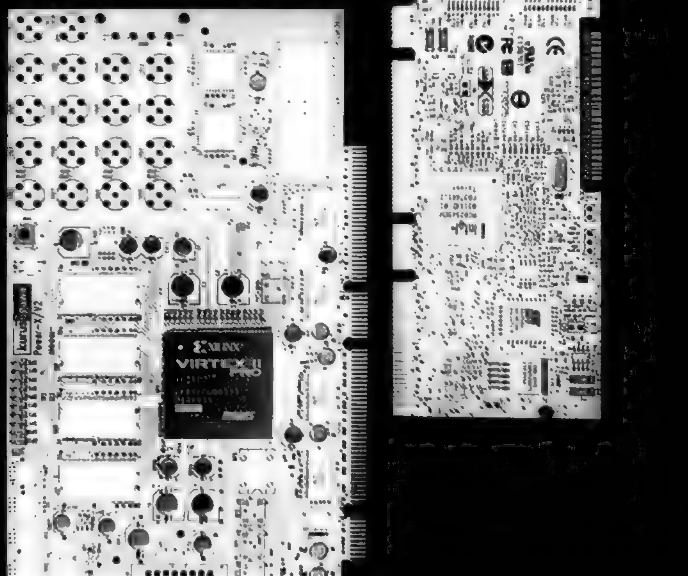
ドライバのピンダウン機能はIRPを完了させない方法で実現しています.この状態は,システムによりIRPの完了が待たれる状態に陥る原因となります.すなわち,プロセスがユーザーによってCtrl-Cなどの方法で終了させられた場合,ロックが解放されない限り,プロセスはIRP待ち状態に陥ります.このような状況を避けるため,必ずピンダウン状態をすべて解放するよう,Ctl-Cなどのイベント用関数をオーバライドしてください.

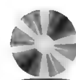
なお,万が一このような状況に陥った場合は,Windowsを再起動する必要があります.

プロセスが自ら完了する際には,ドライバ内で`Cleanup`ディスパッチルーチンが呼ばれるため,解放処理を行っています.この場合は,きちんとロックが解放されるため,プロセスが終了しない状況には陥りません.

参考文献
1) DDKドキュメント
2) WinDbgドキュメント
3) PlatformSDKドキュメント

やまぎわ・しんいち





PCI-PCIブリッジの動作と
組み込み向けPCI BIOSの作成法

PCIバスツリー構造とPCI BIOSの動作

山武 一朗

PC/AT互換機にはPCI BIOSが実装されている．しかし組み込み機器では，汎用的なPCI BIOSは実装されていないことが多い．組み込み機器にPCIバスを採用する場合，各デバイスの初期化をどうするべきだろうか．PCI BIOSを作成するにはPCIバスのツリー構造と各デバイスの初期化方法を理解する必要がある．そこで，ここではPCIバスのツリー構造と各種PCIデバイスのコンフィグレーションレジスタについて解説したあと，PCI BIOSのもっとも重要な仕事であるリソースの割り当てアルゴリズムなどについて解説する．

（編集部）

はじめに

● PCIデバイスはリセット直後は動けない!

第1章からのPCI/PCI-Xの基礎知識で解説したように，PCIデバイスはリセット直後の状態では，コンフィグレーションレジスタに実装された各種イネーブルビットやベースアドレスレジスタがすべてクリアされていて，手も足も出ない状態になっています．これらベースアドレスレジスタに，システムからアドレスを割り当てられ，各種イネーブルビットがセットされてはじめて，メモリやI/Oが使えるようになります．

つまり，PCIデバイスを使うためには必ずリソース割り当てなどの初期化処理が必要になります．PC/AT互換機では，これらの処理をマザーボードのBIOSの中に実装されているPCI BIOSが行っています．

PC/AT互換機では，よほど特殊なPCIデバイスでない限りは，PCIスロットにPCIデバイスを差し込んで電源を入れただけでも，何がしかのアドレスや割り込みが割り当てられた状態でシステムが起動します．このPCI BIOSが，マザーボードのオンボード上からPCI拡張スロットに至るまで，すべてのPCIデバイスを検索し，どのデバイスがどれだけのリソースを必要としているかを調べ，そのシステム内でリソースコンフリクトのないよう各種リソースを割り当てていく処理をします．PC/AT互換機はいわゆるパソコンですから，用途に合わせてさまざまな拡張機器が接続されます．マザーボードに実装されているPCI BIOSは，どんなPCIデバイスが接続されても，可能なかぎりリソースを割り当てるよう努力します．

● PCI BIOSがない!?

しかし同じPCIバスを採用していても，組み込み機器では少し事情が異なります．基本的に組み込み機器は，通常ははじめから特定の用途を想定して設計されていて，汎用的拡張性はありません．またある程度の拡張性があったとしても，物理的なコネクタが独自仕様だったり，汎用拡張スロットと同じコネクタが使われていたとしても，「PCI拡張スロット」とはうたわずに「○○専用拡張スロット」というように，実装可能なボードが限定されているのがほとんどです．

このような状況から，PCIバスを採用した組み込み機器では，どんなデバイスが接続されてもリソースを割り当てようと努力する……というようなPCI BIOSは実装されず，システム初期化ルーチンの中に「このデバイスはこのアドレス，こっちのデバイスの割り込みはこれ」というように，リソースを決め打ちして割り当てる処理が入っている程度です．つまり，知っているデバイスに対してあらかじめ決められたリソースを割り当てるだけで，自分の知らないデバイスが見つかっても，通常は無視（リソースを割り当てない）してしまいます．

● PCI拡張スロットは汎用性を要求される!?

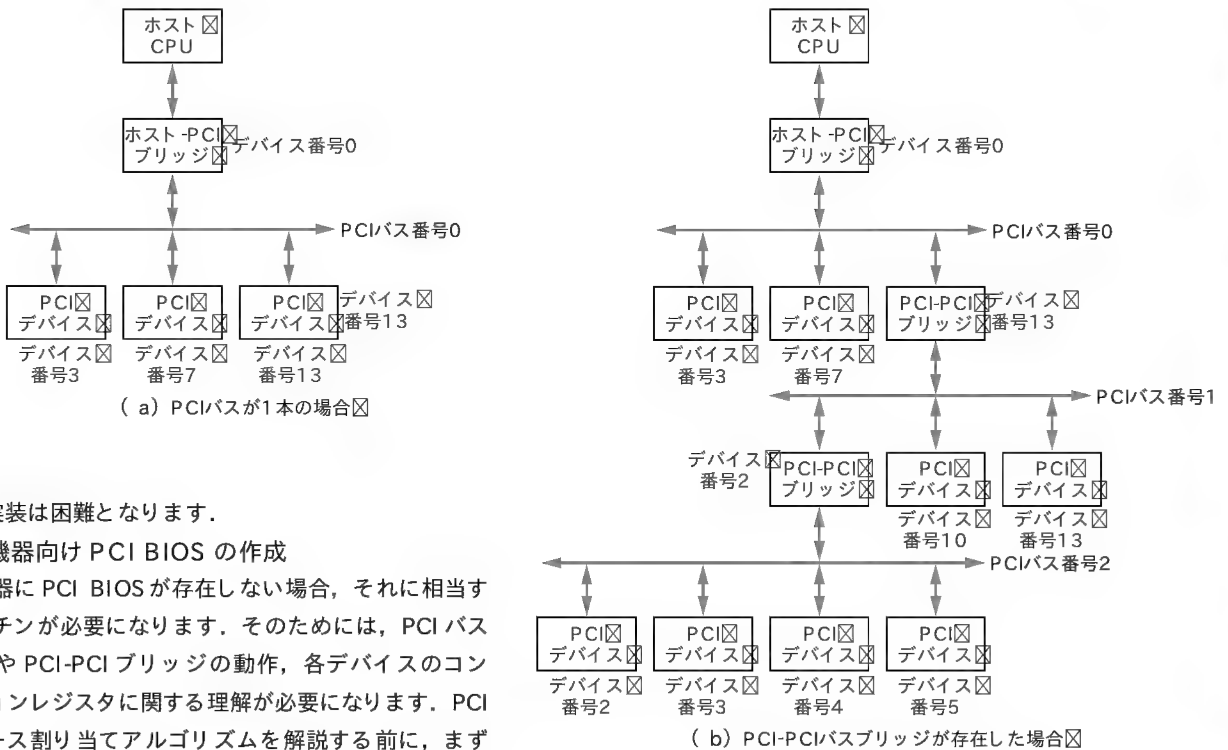
組み込み機器は汎用性を考えていないので，通常はこれで十分なのです．しかしシステムに「PCI拡張スロット」を搭載してしまうと，突然，PC/AT互換機と同様の汎用性が求められてしまいます．

本誌2003年3月号掲載の「CQ RISC評価キット/SH-4PCI with Linux活用研究2 PCIデバイス対応デバイスドライバの作成法」では，まさにその状況が解説されています．このSH-4 CPUボードに実装されているフラッシュROMには，オンボードのPCIデバイスのみを初期化するコードしか記述されておらず，PCI拡張スロットにPCIデバイスを実装しても，リソースが割り当てられないままLinuxカーネルが起動してしまうのです．

● 組み込み向けプロセッサにおけるPCI

また，汎用的なPCI BIOSの実装を難しくしている要因として，組み込み向けプロセッサへのPCIバスの実装方法に問題がある点も挙げられます．Appendix 2でも解説しているように，組み込み向けプロセッサではCPUから見たPCIバス空間が，PC/AT互換機と比較して非常に狭いアドレス空間しかない場合が多いのです．よって，使いもしない（？）デバイスに貴重なアドレス空間を割くことができません．こうなると汎用的な

〔図1〕PCIバスのツリー構造



PCI BIOSの実装は困難となります.

● 組み込み機器向けPCI BIOSの作成

組み込み機器にPCI BIOSが存在しない場合，それに相当する初期化ルーチンが必要になります．そのためには，PCIバスのツリー構造やPCI-PCIブリッジの動作，各デバイスのコンフィグレーションレジスタに関する理解が必要になります．PCI BIOSのリソース割り当てアルゴリズムを解説する前に，まずはこれらの前提となる知識について整理します．

1 PCIバスのツリー構造とブリッジ

● ブリッジを経由したツリー構造

図1(a)に最小構成のPCIバスを示します．ホストCPUの下にホスト-PCIブリッジが存在し，バス番号0のPCIバスが1本だけ存在します．そしてこの1本のPCIバス上に数個のPCIデバイスがぶら下がっています．ここで，もっと多くのPCIデバイスを接続したい場合はどうすればよいでしょうか？　たしかに，PCIの仕様ではデバイス番号は0～31までで32個のデバイスを接続できますが，電気的な負荷を考えるとオンボード実装で6～8個，拡張スロットを使うなら4～6個というのが限界でしょう．

Column

プリフェッチ可能メモリ空間とは

PCIのメモリ空間には，メモリ空間とプリフェッチ可能メモリ空間があります．

PCIメモリ空間に実装するデバイスは，いわゆるメモリデバイスだけでなく，メモリマップトI/Oデバイスを割り当てることもあります．メモリデバイスであれば，ROMであれば書き換わることはなく，RAMであるなら書き換え動作をしない限りは何度読み出しても値が変わることはありません．よって，これらはCPUやイニシエータデバイスがリード要求を出してきたら，あらかじめメモリを先読みしてデータを用意しておくことで，次のリード要求に対して素早く応答することができます．先読みしたデータが不要になったら破棄すればよいだけです．

しかし，メモリマップトI/Oの場合は，一度読み出しただけでステータスがクリアされてしまうようなレジスタが存在するかもしれません．この場合，CPUやイニシエータがアドレスを指定してリード要求を出した時点でアクセスを開始すべきです．

このように，純粋にメモリデバイスとして動作するデバイスであれば，プリフェッチ可能なメモリ空間に割り当てることで，アクセスを高速化することができます．しかしメモリマップトI/Oデバイスをプリフェッチ可能なメモリ空間に割り当ててしまうと，CPUやイニシエータが意図しないアドレスに対してもアクセスが発生してしまうので，動作がおかしくなってしまいます．これらの2種類のメモリ空間は使い分けが必要です．

では，プリフェッチ可能メモリ空間を要求するようベースアドレスレジスタを設定すれば，必ずプリフェッチ可能なメモリ空間に割り当てられるかというと，それはプラットホームに依存します．通常のデスクトップタイプのPC/AT互換機では，プリフェッチ可能ビットを立てても，通常のメモリ空間に割り当ててしまうものがほとんどです．サーバ系のシステムでは，通常メモリ空間とプリフェッチ可能メモリ空間を考えてアドレスを割り当てるものもあるようです．

〔図2〕PC/AT互換機の場合のホストCPUからみたPCIアドレス空間

(a) メモリ空間

(b) I/O空間

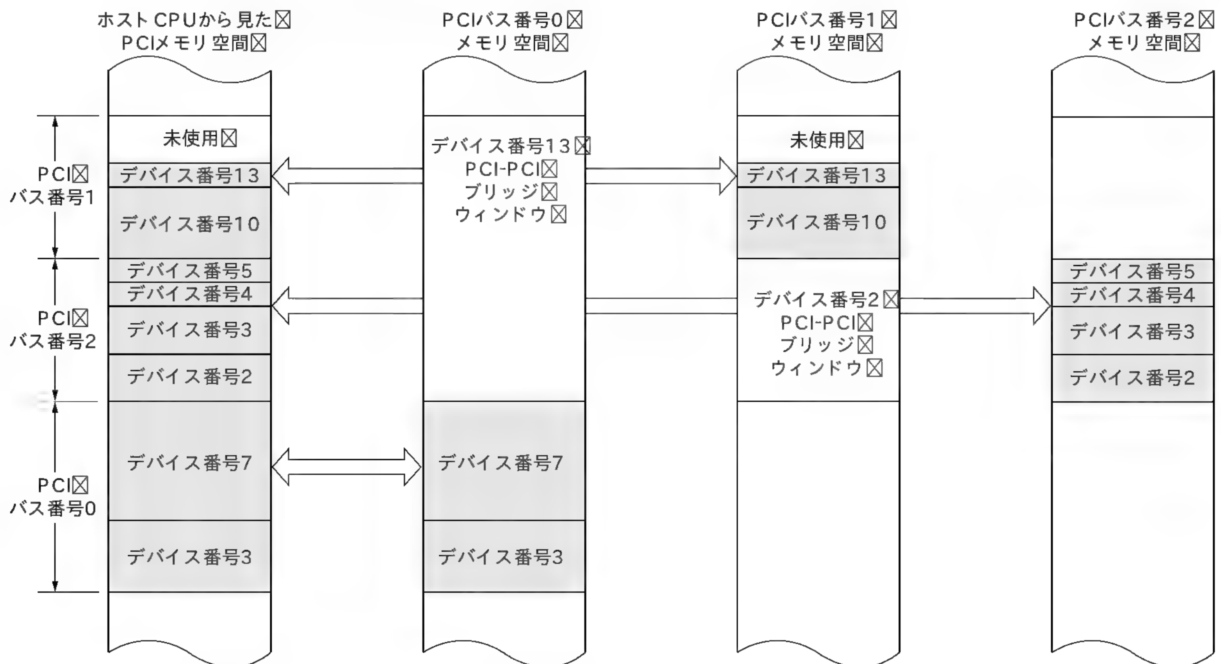
そこでPCIでは，ブリッジという考え方を導入することで，より多くのデバイスを接続可能にしています．図1(b)にPCI-PCIブリッジが存在した場合のPCIバスの構造を示します．PCI-PCIブリッジを介すことで，電気的な限界を超えた個数のPCIデバイスを接続することが可能になります．

ブリッジより上位(ホスト)側はプライマリバス，低位側をセカンダリバスと呼びます．ホスト-PCIブリッジの直下のPCIバスはバス番号0になります．ブリッジを経由した先のPCIバスはバス番号を1，2，…と割り当てていきます．PCIバスの枝分かれのしかたによっては，バス番号の順番はPCI BIOSのバス検索アルゴリズムにより異なる場合もあります．

● ホストCPUから見たPCIアドレス空間

図2に，PC/AT互換機の場合のホストCPUからみたPCIアドレス空間を示します．PCIのメモリ空間は4Gバイトありますが，1Mバイト以下のアドレスには640Kバイトのコンベン

〔図3〕PCI-PCIブリッジによるウィンドウの動作



ショナルメモリやVGA領域，BIOS領域があります．その上にはいわゆるメインメモリが実装されています．メインメモリが実装されていない領域から高位アドレスがPCIメモリ空間となります．I/O空間はもっと複雑で，レガシーI/Oが虫食い状態で割り当てられていて，その隙間をPCIデバイスが使うという形になっています．

ここで，図1のPCIバスのツリー構造と照らし合わせて考えてみましょう．図1(a)のようなPCIバスが1本しかないシステムの場合は，CPUから見えるPCIメモリ空間は，すべてPCIバス番号0のメモリ空間に対応します．I/O空間もしかりです．では，図1(b)のようにPCI-PCIブリッジがあった場合はどうでしょうか．

● 上位バスの一部の空間を下位側へ

PCI-PCIブリッジのもっとも重要な機能は，プライマリバスのアドレス空間の一部をセカンダリバスへ接続することです．これをウィンドウと呼びます．PCI-PCIブリッジのコンフィグレーションレジスタには，このウィンドウの先頭アドレスと終了アドレスを設定するレジスタがあり，アドレスを設定することでブリッジの動作を開始します．

図3に図1(b)の例の場合のPCI-PCIブリッジによるウィンドウの動作を示します．PCI-PCIブリッジが多段に接続されている場合，下位にぶらさがるすべてのデバイスのアドレス空間は，上位のブリッジのウィンドウ内に収まるようにアドレスを設定します．もしベースアドレスレジスタにウィンドウ範囲外のアドレスを設定した場合は，上位バスでそのアドレスに対してバスアクセスを開始しても，下位バスにはそのバスアクセスは伝わりません．

図ではメモリ空間を示していますが，I/O空間でも同様です．

2 コンフィグレーションレジスタのフォーマットについて

次に，PCIデバイスのコンフィグレーションレジスタの構造について確認しておきましょう．

● ヘッダタイプ0 ——通常PCIデバイス

ヘッダタイプ0のデバイスは通常のPCIデバイスです．図4に，通常PCIデバイスのコンフィグレーションレジスタを示します．各レジスタの詳細な意味や動作については，参考文献を参照してください．

標準PCIデバイスについては，6本あるベースアドレスレジスタと，割り込みラインレジスタ，そしてコマンドレジスタのI/O，メモリ，バスマスタの各イネーブルビットをセットすればよいでしょう．

● ヘッダタイプ1 ——標準PCI-PCIブリッジデバイス

ヘッダタイプ1のデバイスは，標準PCI-PCIブリッジデバイスです．図5に標準PCI-PCIブリッジデバイスのコンフィグレーションレジスタを示します．通常PCIデバイスと同様の名称のレジスタは，用途もレジスタ構成も同じです．ここではPCI-PCIブリッジ特有のレジスタについて説明します．

▶ セカンダリステータスレジスタ

アドレス06hのステータスレジスタは，このPCI-PCIブリッジ自身のステータスを示すもので，セカンダリ側のエラーの状

〔図4〕通常PCIデバイスのコンフィグレーションレジスタ(ヘッダタイプ0)

ビット31	ビット16	ビット15	ビット0	オフセット
デバイスID		ベンダID		00h
ステータスレジスタ		コマンドレジスタ		04h
クラスコード			リビジョンID	08h
BIST	ヘッダタイプ	レイテンシタイマ	キャッシュラインサイズ	0Ch
ベースアドレスレジスタ0				10h
ベースアドレスレジスタ1				14h
ベースアドレスレジスタ2				18h
ベースアドレスレジスタ3				1Ch
ベースアドレスレジスタ4				20h
ベースアドレスレジスタ5				24h
ベースアドレスレジスタ6				28h
サブシステムID		サブシステムベンダID		2Ch
拡張ROMベースアドレスレジスタ				30h
予約			新機能ポインタ	34h
最大レイテンシ	最小グラント	割り込みピン	割り込みライン	3Ch
ベンダ定義				40h～FFh

(a) コンフィグレーションレジスタ一覧



(b) コマンドレジスタ

31(または63)　4 3 2 1 0

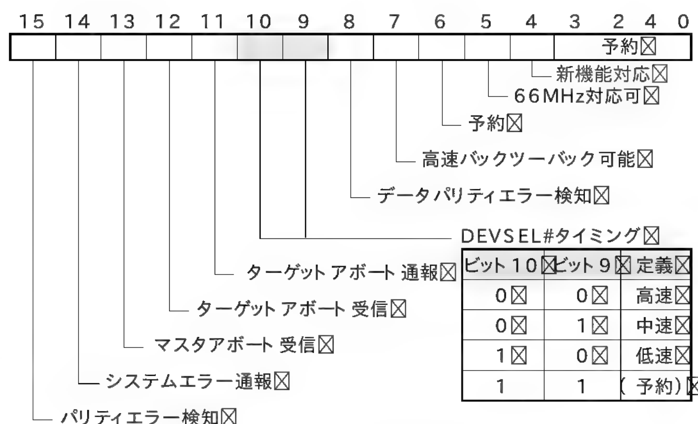
ベースアドレス　プリフェッチ　タイプ　メモリ空間インジケータ

	プリフェッチ
1	可
0	不可

ビット2	ビット1	定　義
0	0	32ビット空間の任意の位置
0	1	1Mバイト以下のメモリ空間[注]
1	0	64ビットアドレス空間の任意の位置
1	1	予約

注：PCIバス仕様Rev.2.2から予約

(d) メモリ空間用ベースアドレスレジスタ



(c) ステータスレジスタ

31　2 1 0

ベースアドレス　予約　I/O空間インジケータ

(e) I/O空間用ベースアドレスレジスタ

況などを示すものではありません．セカンダリ側のエラーの状況などを示すレジスタとして，セカンダリステータスレジスタが定義されています．基本的には通常のステータスレジスタと同じレジスタフォーマットです．

▶プライマリ/セカンダリ/サブボーディネートバス番号

PCI-PCIブリッジは，文字どおり上位のPCIバスと下位のPCIバスをブリッジするデバイスです．ブリッジの上位のバスをプライマリバス，下位のバスをセカンダリバスと呼びます．プライマリバス番号には，PCI-PCIブリッジ自身が接続されているPCIバス番号を，セカンダリバス番号にはPCI-PCIブリッジの下に接続されるPCIバスの番号を格納します．

もう一つ聞き慣れないバス番号として，サブボーディネートバス番号があります．これは，PCI-PCIブリッジが多段に接続された場合，セカンダリバス番号には自身のすぐ下にくるバス番号を入れますが，そのバス上にPCI-PCIブリッジが存在する場合には，セカンダリバス番号よりさらに大きな番号をもつPCIバスが存在することになります．上位のPCI-PCIブリッジは，自分の下にバス番号何番までのPCIバスが存在するのかを把握している必要があります．サブボーディネートバス番号とは，この番号を格納するレジスタです．

▶セカンダリレイテンシタイマ

アドレス0Dhのレイテンシタイマは，このPCI-PCIブリッジ自身のレイテンシタイマを示すもので，セカンダリ側のレイテンシタイマを設定するものではありません．ステータスレジスタ同様に，セカンダリ側のレイテンシタイマを設定します．

▶I/Oおよびメモリのベースアドレス/リミットアドレス

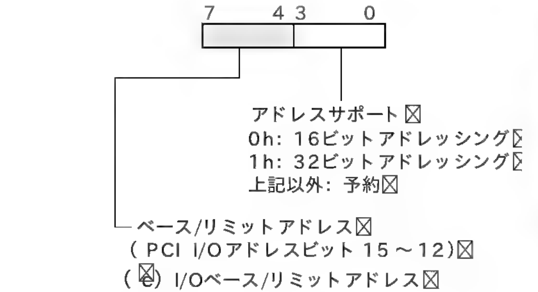
ブリッジの役割でもっとも重要な機能を設定するレジスタです．

I/Oベースアドレスは I/Oウィンドウの先頭アドレスを，I/Oリミットアドレスは I/Oウィンドウの終了アドレスを設定します．8ビットのうち下位4ビットで16ビットアドレッシングか32ビットアドレッシングかを指定します．32ビットアドレッシングが指定されている場合は，I/Oベースアドレス(上位16ビッ

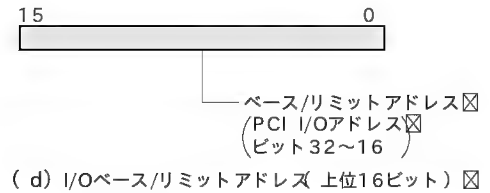
〔図5〕標準PCI-PCIブリッジデバイスのコンフィグレーションレジスタ（ヘッダタイプ1）

ビット31	ビット16	ビット15	ビット0	オフセット
デバイスID		ベンダID		00h
ステータスレジスタ		コマンドレジスタ		04h
クラスコード			リビジョンID	08h
BIST	ヘッダタイプ	プライマリレイテンシタイマ	キャッシュラインサイズ	0Ch
ベースアドレスレジスタ0（オプション）				10h
ベースアドレスレジスタ0（オプション）				14h
セカンダリレイテンシタイマ	サブボーディネートバス番号	セカンダリバス番号	プライマリバス番号	18h
セカンダリステータスレジスタ		I/Oリミットアドレス	I/Oベースアドレス	1Ch
メモリリミットアドレス		メモリベースアドレス		20h
プリフェッチ可能メモリリミットアドレス		プリフェッチ可能メモリベースアドレス		24h
プリフェッチ可能メモリベースアドレス（上位32ビット）				28h
プリフェッチ可能メモリリミットアドレス（上位32ビット）				2Ch
I/Oリミットアドレス（上位16ビット）		I/Oベースアドレス（上位16ビット）		30h
予約			新機能ポインタ	34h
拡張ROMベースアドレスレジスタ（オプション）				30h
ブリッジコントロール		割り込みピン	割り込みライン	3Ch

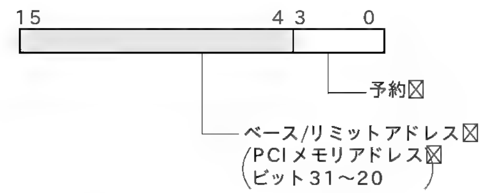
（a）レジスタ一覧



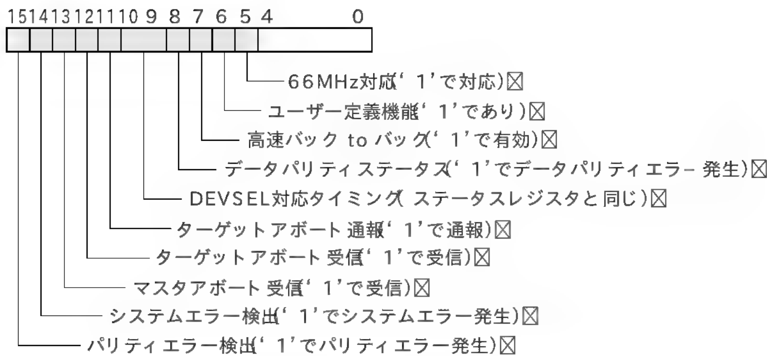
（c）I/Oベース/リミットアドレス



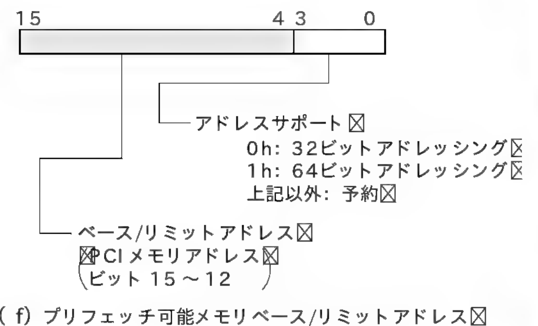
（d）I/Oベース/リミットアドレス（上位16ビット）



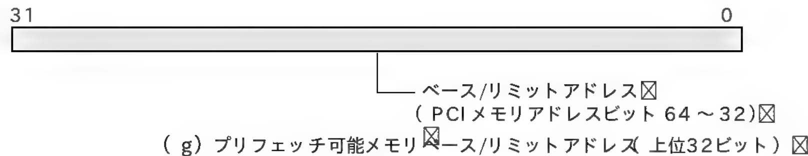
（e）メモリベース/リミットアドレス



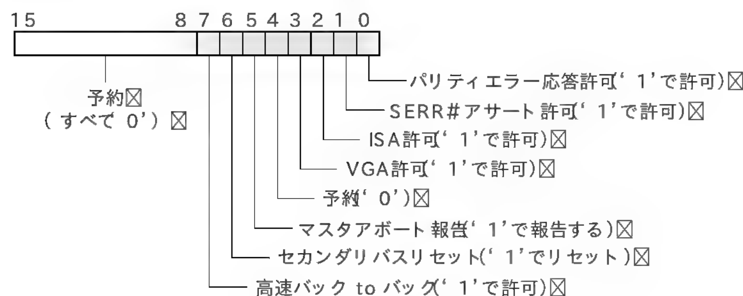
（b）セカンダリステータスレジスタ



（f）プリフェッチ可能メモリベース/リミットアドレス



（g）プリフェッチ可能メモリベース/リミットアドレス（上位32ビット）



（h）ブリッジコントロール

ト）およびI/Oリミットアドレス（上位16ビット）も設定します．上位4ビットがPCI I/Oアドレスのビット15～12に相当します．PC/AT互換機をはじめとして，一般的にはI/O空間は16ビットアドレッシングで使うので，たとえばI/Oベースアドレスに20h，I/Oリミットアドレスに20hを設定すると，2000h～2FFFhまでのPCI I/Oアドレス空間がセカンダリバス側につながるようになります．

通常メモリ空間およびプリフェッチ可能メモリ空間の場合も同様で，メモリベースアドレスがメモリウィンドウの先頭アドレスを，メモリリミットアドレスがメモリウィンドウの終了アドレスを指定します．違いは，通常メモリ空間は32ビットアドレッシングのみで，プリフェッチ可能メモリ空間は32ビット/64ビットアドレッシングの2種類がある点です．

通常メモリ空間のメモリベース/リミットアドレスの下位4ビットは予約です（32ビットアドレッシングしかないため）．プリフェッチ可能メモリ空間のメモリベース/リミットアドレスの下位4ビットは，32ビットアドレッシングか64ビットアドレッシングかを指定します．64ビットアドレッシングが指定されている場合は，プリフェッチ可能メモリベースアドレス（上位32ビット）およびプリフェッチ可能メモリリミットアドレス（上位32ビット）も設定します．ビット15～4まではPCIメモリアドレスのビット31～20に相当します．たとえばメモリベースアドレスに0100h，メモリリミットアドレスに01F0hを設定すると，01000000h～01FFFFFFhまでのPCIメモリ空間がセカンダリバス側につながるようになります．

また，I/Oおよびメモリともに，ベースアドレスよりリミットアドレスの値を大きな値に設定すると，そのウィンドウはディセーブル状態になります．

▶ブリッジ制御レジスタ

セカンダリバス側のリセット信号やエラー/アボートなどの許可設定などを行います．

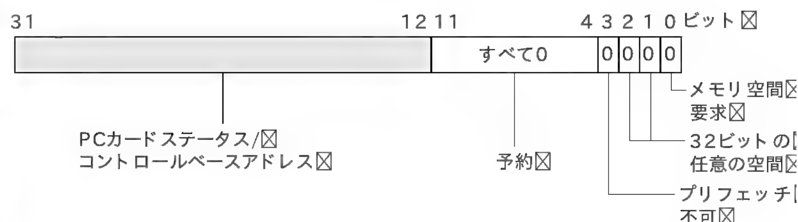
● ヘッダタイプ2 —— 標準PCI-CardBusブリッジデバイス

ヘッダタイプ2のデバイスは標準PCI-CardBusブリッジデバイスです．図6に標準PCI-CardBusブリッジデバイスのコンフィグレーションレジスタを示します．

〔図6〕標準PCI-CardBusブリッジデバイスのコンフィグレーションレジスタ（ヘッダタイプ2）

ビット31	ビット16	ビット15	ビット0	オフセット
デバイスID		ベンダID		00h
ステータスレジスタ		コマンドレジスタ		04h
クラスコード			リビジョンID	08h
BIST	ヘッダタイプ	レイテンシタイマ	キャッシュラインサイズ	0Ch
PCカードステータス/コントロールベースアドレス				10h
セカンダリステータス		予約	新機能ポインタ	14h
レイテンシタイマ	サブボーディネートバス番号	CardBusバス番号	PCIバス番号	18h
メモリベースアドレスレジスタ0				1Ch
メモリリミットアドレスレジスタ0				20h
メモリベースアドレスレジスタ1				24h
メモリリミットアドレスレジスタ1				28h
I/Oベースアドレスレジスタ0				2Ch
I/Oリミットアドレスレジスタ0				30h
I/Oベースアドレスレジスタ1				34h
I/Oリミットアドレスレジスタ1				38h
ブリッジコントロール		割り込みピン	割り込みライン	3Ch
サブシステムID		サブシステムベンダID		40h
16ビットPCカード レガシーモードベースアドレス				44h
ベンダ定義				48h～FFh

（a）コンフィグレーションレジスタ一覧



（b）PCカードステータス/コントロールベースアドレスレジスタ

CardBusを含むPCカードは，システム稼動状態でカードを抜き差しできるシステムです．一般的なPCI BIOSでは，CardBusブリッジ自身が使うリソースについての初期化のみを行い，CardBusブリッジの先に接続されるPCカードの初期化までは行いません．PCカードのそのものの検出や初期化には，一般的にはカードイネーブラと呼ばれるソフトウェアを使用します．

CardBusブリッジ自身が使うリソースとしては，PCカードの抜き差しの判定や電源/リセット制御などを行う制御レジスタと，カード抜き差しや電源制御など各種カード制御イベントを割り込み制御で駆動できるよう，割り込みも1ソケットあたり1本使います．PCI BIOSとしては，アドレス10hのPCカードステータス/コントロールベースアドレスとアドレス3Chの割り込みラインレジスタなどを初期化します．

● PCIブリッジの変形版

PentiumⅡ世代から登場したグラフィックス専用のAGPも，じつは内部的にはPCIバスとして扱われています．さらに最新のインテル系チップセットでは，チップセット間の通信バスとしてハブアーキテクチャが採用されていますが，これらも論理的にはPCIバスと同等の扱いとなります（図7）．具体的には，AGPデバイスやハブアーキテクチャのデバイスに対しても，リソースの割り当て方法やデバイス

ドライバの初期化手順などには，これまでのPCIデバイスと同様の手法を使うことができます．

● その他のブリッジ系デバイス

PCIの仕様では，PCI-PCIブリッジやPCI-CardBusブリッジ以外にもいくつかのブリッジが規定されています．PCIバスツリー構造のもっとも上位にあるホスト-PCIブリッジ，PC/AT互換機などでレガシー系I/Oを実装するISAバスブリッジなどがあります．しかしこれらは，たとえクラスコードがブリッジデバイスであることを示していても，ヘッダタイプ0のコンフィグレーションレジスタを実装しています．

PCI BIOSではヘッダタイプが0だった場合は，クラスコードでブリッジデバイスが示されていたとしても，通常PCIデバイスと同様の初期化手順を適用します．

3 PCI BIOSの初期化手順

PCIバスのツリー構造や各PCIデバイスおよびブリッジのコンフィグレーションレジスタ構造について理解したところで，次はPCI BIOSがデバイスを初期化していく手順や，ベースアドレスレジスタへのリソース割り当てアルゴリズムなどについて解説します．

ここではもっとも一般的なシステム構成を想定し，メモリ空間は32ビットアドレッシングのみ，またヘッダタイプ0とヘッダタイプ1のデバイスのみを初期化してゆく方法について解説します．

● 基本的なバスの検索順序

基本的な処理の流れとしては，まずはじめにバス番号0上でデバイス番号0から31まで順番にPCIデバイスが存在するかを調べていきます．デバイスを発見した場合はヘッダタイプを確認し，PCI-PCIブリッジである場合は，先にそのブリッジの下を検索するようにします．このとき，検索するバス番号を引き数として，デバイス検索ルーチンを関数として作成すると，C言語では再帰呼び出しが使えるので便利です．

PCI-PCIブリッジの存在しないPCIバスまできたら，そこから最終的なリソースを割り当て関数を抜けていくという流れになります．

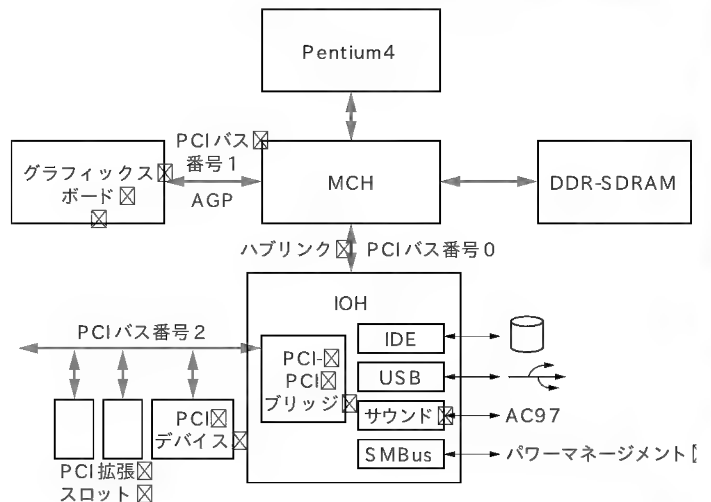
なお，バス番号0/デバイス番号0/ファンクション番号0は通常はホスト-PCIブリッジになります．

● PCI-PCIブリッジの暫定初期化

PCIバスリセット直後は，PCI-PCIブリッジもすべてがディセーブル状態なので，コンフィグレーションサイクルを発行しても，ホスト-PCIブリッジ直下のPCIバス番号0のバス上しかコンフィグレーションサイクルが発行されません．そこでPCIデバイスの検索中にPCI-PCIブリッジを発見した場合は，暫定的にリスト1のⒶのようにブリッジを初期化します．

まず，プライマリバス番号に現在検索中のバス番号を書き込みます．また，それまでに検索したPCIバス番号で最大のバス

〔図7〕ハブアーキテクチャおよびAGP搭載のマシンのPCIバス構造



番号を＋1して保持すると同時に，セカンダリバス番号にもその値を書き込みます．さらにサブボーディネートバス番号には暫定的にバス番号255を設定します．これで，セカンダリバス番号でコンフィグレーションサイクルを発行すると，そのPCI-PCIブリッジの下のバスにコンフィグレーションサイクルが伝わるようになります．

● デバイスの存在確認

あるPCIバス上で，デバイス番号0～31にデバイスが存在するかどうかを検索する処理の流れについて説明します．

（1）PCIデバイスの存在確認

まずはデバイス番号0/ファンクション番号0の，レジスタアドレス0のコンフィグレーションレジスタを32ビット幅でリードします．そこにデバイスが存在しなければ，FFFF_FFFFhが読み出されます．デバイスが存在しなければ，デバイス番号を＋1して次のデバイスの検索に移ります．何らかのデバイスが存在する場合は，次の処理に移ります（リスト2ⓐ）．

（2）シングル/マルチファンクションの判定

デバイスの存在が確認された場合，次はそのデバイスがシングルファンクションデバイスかマルチファンクションデバイスかを判定します．

シングルファンクションデバイスの場合，ファンクション番号1以降にはファンクション番号0のデバイスのコンフィグレーションレジスタが見えてしまう（ファンクション番号をフルデコードしていない）PCIデバイスが存在します．そのため，そのデバイスが本当にシングルファンクションデバイスかマルチファンクションデバイスかを判定する必要があり，ファンクション番号0のヘッダタイプレジスタを読み出し，ビット7が‘0’ならシングルファンクションデバイス，‘1’ならマルチファンクションデバイスであると判定します（リスト2ⓑ）．

シングルファンクションデバイスであると判定した場合は，

〔リスト1〕PCI-PCIブリッジの初期化

```
/* PCI標準バスブリッジ発見時の前処理と後処理 */
int PCIBIOS_InitPCI2PCI(int BusNo,int DevNo,int FuncNo,int INTLineP)
{
    int reslt;
    ULONG data;
    PCI_MaxBusNo++;     /* PCIバス番号インクリメント */
    /* 暫定的にバス番号255まで応答 */
    data=(255<<16) | ((PCI_MaxBusNo-1)<<8) | BusNo;
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x18,data);
    data=(PCI_IoAddress-1)  & 0xf000;      /* I/Oリミットアドレス */
    PCIBridge_CfgWordWrite(BusNo,DevNo,FuncNo,0x1c,data);
    data=(PCI_MemAddress-1) & 0xfff00000;  /* メモリリミットアドレス */
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x20,data);
    /* プリフェッチメモリリミットアドレス */
    data=(PCI_PreAddress-1) & 0xfff00000;
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x24,data);
    /* Upper32ビットメモリ領域はゼロクリア */
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x28,0);
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x2c,0);
    /* Upper16ビットI/O領域はゼロクリア */
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x30,0);
    INTLineP=INTLineP+(DevNo & 3);
    if (INTLineP > 3) INTLineP=INTLineP-4;

    /* 下のバスを検索(再帰呼び出し) */
    reslt=PCIBIOS_InitSub(PCI_MaxBusNo-1,INTLineP);

    /* ブリッジの下にI/O空間を使用するデバイスがいる */
    if (reslt & 1) {
        /* I/Oウィンドウを設定する */
        data=PCIBridge_CfgWordRead(BusNo,DevNo,FuncNo,0x1c);
        data=(data & 0xf000) | ((PCI_IoAddress >> 8) & 0xff);
        PCIBridge_CfgWordWrite(BusNo,DevNo,FuncNo,0x1c,data);
    } else {  /* ブリッジの下にI/O空間を使用するデバイスはいない */
        /* リミット<ベース→I/Oウィンドウは開かない */
        PCIBridge_CfgWordWrite(BusNo,DevNo,FuncNo,0x1c,0x000000ff);
    }

    /* ブリッジの下に通常メモリ空間を使用するデバイスがいる */
    if (reslt & 2) {
        /* メモリウィンドウを設定する */
        data=PCIBridge_CfgLongRead(BusNo,DevNo,FuncNo,0x20);
        data=(data & 0xfff00000) | (PCI_MemAddress >> 16);
        PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x20,data);
    } else {  /* ブリッジの下にメモリ空間を使用するデバイスはいない */
        /* リミット<ベース→メモリウィンドウは開かない */
        PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x20,0x0000ffff);
    }

    /* ブリッジの下にプリフェッチメモリ空間を使用するデバイスがいる */
    if (reslt & 4) {
        /* プリフェッチメモリウィンドウを設定する */
        data=PCIBridge_CfgLongRead(BusNo,DevNo,FuncNo,0x24);
        data=(data & 0xfff00000) | (PCI_PreAddress >> 16);
        PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x24,data);
    } else {
    /* ブリッジの下にプリフェッチメモリ空間を使用するデバイスはいない */
        /* リミット<ベース→プリフェッチメモリウィンドウは開かない */
        PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x24,0x0000ffff);
    }

    /* ブリッジの下にデバイスが存在しなくても共通で行う処理 */
    data=PCIBridge_CfgLongRead(BusNo,DevNo,FuncNo,0x18);
    /* 下に繋がっているバスの最大バス番号設定 */
    data=(PCI_LatencyTimer<<24)|((PCI_MaxBusNo-1)<<16)|(data & 0xffff);
    PCIBridge_CfgLongWrite(BusNo,DevNo,FuncNo,0x18,data);
    /* セカンダリステータスレジスタクリア */
    PCIBridge_CfgWordWrite(BusNo,DevNo,FuncNo,0x1e,0xffff);
    /*ブリッジコントローラクリア */
    PCIBridge_CfgWordWrite(BusNo,DevNo,FuncNo,0x3e,0);

    return reslt;
/* ビット0:I/O使用 ビット1:メモリ使用 ビット2:プリフェッチメモリ使用 */
}
```


ファンクション番号1～7にはアクセスしません．これらのファンクション番号のデバイスにアクセスして何らかのコンフィグレーションレジスタが読み出せたとしても，それを無視するようにします．

マルチファンクションデバイスであると判定した場合は，同一のデバイス番号で，ファンクション番号1～7にもアクセスして，そこにデバイスが存在するかを確認します．ファンクション番号1～7がすべて使われていることはあまりありません．二つだけの場合もあります．ここで注意が必要なのは，ファンクション番号が歯抜けで使われている場合があることです．たとえ二つしかファンクションが実装されていなくても，ファンクション番号として0と3が使われていて，1と2が使われない場合もあるのです．よって，ファンクション番号を順番に検索していき，FFFF_FFFFhが読み出せた時点でマルチファンクションデバイスの検索を打ち切るという処理では，歯抜けになった番号の後にあるデバイスを見落としてしまうことになります．

また，シングルファンクションであろうとマルチファンクションであろうと，ファンクション番号0は必ず存在するということになります．

(3) ヘッダタイプの判定

シングルファンクションかマルチファンクションかの判定でヘッダタイプを読み出しましたが，このレジスタの下位7ビットには，そのPCIデバイスのコンフィグレーションレジスタフォーマットが示されています．0だった場合は通常のPCIデバイス，1だった場合はPCI-PCIブリッジ，2だった場合はPCI-CardBusブリッジとなります(リスト3)．

ここではヘッダタイプ0と1のみを扱うので，ヘッダタイプ2のPCI-CardBusブリッジの場合は，そのデバイスを無視して次のデバイス/ファンクションを検索します．PCI-PCIブリッジの場合は，PCI-PCIブリッジの暫定初期化で説明した手順でブリッジを初期化し，デバイス検索関数を再帰呼び出しして一つ下のPCIバスを同様に検索することになります．ヘッダタイプ0の通常のPCIデバイスだった場合は，次の処理に移ります．

(4) ベースアドレスレジスタの確認

ヘッダタイプ0のPCIデバイスを発見した

〔リスト 2〕PCIデバイスの存在確認

```
for(l=0;l<32;l++){/* デバイス番号0～31検索 */
    /* ファンクション番号0/レジスタ0をリード */
    data=PCIBridge_CfgLongRead(PCI_BusNo,l,0,0);
    if (data != 0xffffffff) {  /* 何かデバイスがいる!! */ ←── Ⓐ
        /* ファンクション番号0/ヘッダタイプをリード */
        HeadType=PCIBridge_CfgByteRead(PCI_BusNo,l,0,0x0e);
        if (HeadType&0x80) {  /* マルチファンファンクション */ ┐
            FuncNo_End=8;                                      │Ⓑ
        } else {              /* シングルファンクション */     │
            FuncNo_End=1;                                      ┘
        }
        for(j=0;j<FuncNo_End;j++){
        /* 最終ファンクション番号までチェック */

             リスト 3が入る
                  リスト 4が入る

        }
    }
}
```

〔リスト 3〕ヘッダタイプの判定

```
BadDevice=0; /* リソース割り当て不能デバイス フラグクリア */

if ((HeadType&0x7f)==1) {   /* ヘッダタイプ1(PCI-PCIブリッジ) */
    /* クラスコードチェック */
    data=PCIBridge_CfgLongRead(PCI_BusNo,l,j,8) >> 8;
    if (data == 0x060400) { /* PCI標準バスブリッジ発見!! */
        /* PCI-PCIブリッジの下を検索 */
        reslt=reslt |
                    PCIBIOS_InitPCI2PCI(PCI_BusNo,l,j,INTLineP);
    } else {  /* ヘッダタイプ1でPCI標準バスブリッジ以外のデバイス */
        /* リソース割り当て不能デバイスだった場合のフラグセット */
        BadDevice=1;
    }

} else if ((HeadType&0x7f)==2) { /* ヘッダタイプ2(CardBusブリッジ) */
    /* CardBusコントローラには未対応 */
    BadDevice=1;

} else { /* ヘッダタイプ0(通常デバイス) */

    リスト 4が入る

}
```

Column 2

ベースアドレスレジスタが1本も使われていないPCIデバイス!?

PCIデバイスによっては，ベースアドレスレジスタが1本も使われていないデバイスもありえます．つまり，I/Oもメモリも何も要求しない/使わないというデバイスです．そんなデバイスがPCIバス上に存在する意味があるのか？と思われますが，一般的にこのようなデバイスは，じつはシステムとして重要な役割を担っている場合がよくあります．

たとえばリストAを見てください．これはあるPC/AT互換機で実装されているPCIデバイスですが，ベースアドレスレジスタがすべてゼロとなっています．デバイスの名称を見ると，「PowerManagement Controller」と表示されています．

このようなデバイスは，そのシステムに固有の方法で電源制御などを行うため，通常のPCIデバイスのようにI/O空間やメモリ空間を要求する必要がないためです．

この他に，マザーボードのBIOSだけが存在を知っているデバイス，俗に言う「隠しデバイス」というものもあります．たとえば，ファンクション番号0でアクセスしてもすべてFFFF_FFFFhが読み出せるため，これまで説明してきたデバイス検索アルゴリズムでは「デバイスは存在しない」で終わってしまうところですが，じつはファンクション番号7でアクセスすると，PCIデバイスが存在するというものです．

これは，マザーボード上にそのシステムで独自仕様のデバイスが実装されており，マザーボードのBIOSだけがこのデバイスの機能を使うような場合に採用されます．

〔リスト A〕ベースアドレスを要求しないPCIデバイス

```
Bus Dev Fnc VendorID DevID Rev
  0   7   3    8086h  7113h 02h Intel 82371AB PIIX4 Power management Controller
Vendor ID            W(00h) 8086h [Intel]
Device ID            W(02h) 7113h [82371AB PIIX4 Power management Controller]
Command              W(04h) 0003h ['0000 0000 0000 0011'b]
Status               W(06h) 0280h ['0000 0010 1000 0000'b]
Revision ID          B(08h) 02h
Program I/F          B(09h) 00h          Subclass Code         B(0Ah) 80h
Base Class Code      B(0Bh) 06h [Other Bridge Device]
Cache Line Size      B(0Ch) 00h          Latency Timer         B(0Dh) 00h
Header Type          B(0Eh) 00h          BIST                  B(0Fh) 00h
Base Address Reg.0,1 D(10h) 00000000h    D(14h) 00000000h
Base Address Reg.2,3 D(18h) 00000000h    D(1Ch) 00000000h ← ベースアドレスが
Base Address Reg.4,5 D(20h) 00000000h    D(24h) 00000000h   1本も要求されていない
CardBus CIS          D(28h) 00000000h
SubSystem Vendor ID  W(2Ch) 0000h
SubSystem Device ID  W(2Eh) 0000h
Expantion ROM Address D(30h) 00000000h   CAP_PTR               B(34h) 00h
Reserved             T(35h) 000000h      D(38h) 00000000h
Interrupt Line       B(3Ch) 00h          Interrupt Pin         B(3Dh) 00h
Minimum Grant        B(3Eh) 00h          Maximum Latency       B(3Fh) 00h
```

ら，合計6本あるベースアドレスレジスタのうちどれが使われているか，またそれぞれのベースアドレスレジスタが，I/O空間を要求しているのか，メモリ空間を要求しているのか，そしてその空間サイズはいくらかを調べる必要があります．

ベースアドレスレジスタはコンフィグレーションレジスタアドレス10hから27hまでに割り当てられています．これもファンクション番号同様，一つしか使われない場合もあれば，複数使われている場合もあり，さらにベースアドレスレジスタの番号が歯抜けで使われている場合もあります．なおファンクション番号と違う点は，ベースアドレスレジスタ0が未使用である場合や，まったく使われていない場合もあることです(コラム2)．

よってここでは，ベースアドレスレジスタが何本使われているかといった判定はせずに，ベースアドレスレジスタ0～5までをすべてチェックします．

(5) 各ベースアドレスレジスタの要求空間サイズの取得

ベースアドレスレジスタを読み出し，すべてのビットがゼロであっても，そのベースアドレスレジスタが未使用であるとは判断できません．I/O空間を要求するベースアドレスレジスタはビット0に'1'が立っているのですぐに判定できますが，通常の32ビットメモリ空間(非プリフェッチメモリ空間)を要求するベースアドレスレジスタは，下位4ビットもすべてゼロだからです．

そこで，ベースアドレスレジスタにFFFF_FFFFhを書き込んでから読み出し，読み出した値がゼロ以外であれば，そのベースアドレスレジスタは何らかのアドレス空間を要求していると判断します(リスト4Ⓐ)．

また，この時点でベースアドレスレジスタが要求する空間サイズも判定できます．たとえば16Mバイトのメモリ空間を要求する場合は，ベースアドレスレジスタのビット31～24までに書き換え可能なビットを実装します．よってこのベースアドレスレジスタにFFFF_FFFFhを書き込んでから読み出すと，FF00_0000hが読み出されるはずです．下位ビット側から'0'であるのビットを数えると24ビット分となり，2の24乗＝16Mバイトということがわかるわけです．

正確には，メモリ空間を要求するベースアドレスレジスタの下位4ビットは別の意味で使われているので，下位4ビットは強制的にゼロであると見なしてビットをサーチしてください．

また，I/O空間を要求するベースアドレスレジスタの場合，ベースアドレスレジスタのビット31～16まではゼロ固定である場合が

Column 3

拡張ROMベースアドレスレジスタは使わない？

世の中に流通しているPCI拡張ボードのうち，拡張ROMを実装しているPCIボードのほとんどが，PC/AT互換機用の拡張BIOSを実装していると考えて間違いないでしょう．つまり，拡張ROMの中に実装されている拡張BIOSは，x86系用のコードが格納されているわけです．

組み込み機器として，PC/AT互換機アーキテクチャを採用するならいざ知らず，非x86系CPUを搭載した組み込みシステムの場合は，まずほとんどの場合，拡張ROMは不要なリソースでしかありません．よってここで想定するPCI BIOSでも，このレジスタは完全に無視し，まったくリソースを割り当てていません．

〔リスト4〕ベースアドレスレジスタの確認

```
for(i=0;i<=5;i++){/* ベースアドレスレジスタチェック */
    /* All FFhを書き込む */
    PCIBridge_CfgLongWrite(PCI_BusNo,l,j,0x10+(i*4),0xffffffff);
    /* 要求サイズをチェック */
    data=PCIBridge_CfgLongRead(PCI_BusNo,l,j,0x10+(i*4));
    if (data != 0) {    /* レジスタが実装されている */

         if (data & 1) {     /* I/O空間を要求 */
              if ((data & 0x0000ff00) != 0xff00) {
                   /* 256バイトを超えるI/O空間要求は異常 */
                   BadDevice=1;
              } else {  /* 256バイト以下を要求 */
                   BaseIo[IoCount].PCIDevAdr=
                        PCIBIOS_BusDevFunc(PCI_BusNo,l,j,0x10+(i*4));
                   BaseIo[IoCount].BaseAdr=data & 0x0000fffc;
                   IoCount++;
              }

         } else {  /* メモリ空間を要求 */
              if ((data & 6) == 4) {  /* 64ビットアドレス空間を要求 */
                   /* リソース割り当て不能デバイスだった場合フラグセット */
                   BadDevice=1;
                   i++; /* 次のベースアドレスレジスタとペアで使用されるので次もスキップ */
              /* 1Mバイト未満のアドレス空間を要求 */
              } else if ((data & 6) == 2) {
                   /* リソース割り当て不能デバイスだった場合フラグセット */
                   BadDevice=1;
              /* 32ビットアドレスを使用するデバイス */
              } else {
                   /* 256Mバイトを超える空間を要求 */
                   if ((data & 0xf0000000) != 0xf0000000) {
                        /* 256Mバイトを超える空間を要求するデバイスは未対応 */
                        BadDevice=1;
                   /* リソース割り当て可能なデバイス */
                   } else if ((data & 8) == 8) {     /* プリフェッチ可能 */
                        BasePre[PreCount].PCIDevAdr=
                                  PCIBIOS_BusDevFunc(PCI_BusNo,l,j,0x10+(i*4));
                        BasePre[PreCount].BaseAdr=data & 0xfffffff0;
                        PreCount++;
                   } else {  /* 通常メモリ空間使用デバイス */
                        BaseMem[MemCount].PCIDevAdr=
                                  PCIBIOS_BusDevFunc(PCI_BusNo,l,j,0x10+(i*4));
                        BaseMem[MemCount].BaseAdr=data & 0xfffffff0;
                        MemCount++;
                   }
              }
         }
    }
}
```

あります．PC/AT互換機ではI/O空間が64Kバイトしかないため，下位16ビットのみに書き換え可能なビットを実装した設計になっているPCIデバイスが存在するためです．上位16ビットがゼロ固定であっても，下位側（ビット2）から'0'であるビットをサーチしてI/O空間のサイズを判定できます．

なお，メモリ空間を要求するベースアドレスレジスタでは，64ビットアドレッシング空間を要求してくる場合もあります．ここでは32ビットアドレッシングのみを考えているので，64ビットアドレッシングを要求してくるベースアドレスレジスタは無視してください．ただしこの場合，次のベースアドレスレジスタは上位32ビット分のベースアドレスレジスタになるので，一つレジスタを飛ばす処理を入れてください．

さらに，1Mバイト未満のアドレスや，256Mバイトを超える空間を要求するデバイスも無視することにしています．

（6）全デバイス番号/全ファンクション番号を検索

以上のような検索で，デバイス番号0～31まで，マルチファンクションデバイスの場合はファンクション番号1～7も検索し，それぞれのデバイスのどのベースアドレスがどれくらいのアドレス空間を要求しているかの一覧情報をリストアップしたら終了です．

●リソース割り当て処理

同一PCIバス上でのデバイス検索が終了したら，その中でリソースを割り当てていくわけですが，ここで少し工夫が必要です．

図4（d）で示したベースアドレスレジスタの構造でわかるように，PCIデバイスでは2のn乗単位でアドレス空間を要求します．また，割り当て可能なアドレスも要求空間単位の先頭アドレスとなります．たとえば，16Mバイトのメモリ空間を要求するベースアドレスレジスタには，16Mバイト単位の先頭アドレスしか割り当てられません．ベースアドレスレジスタのうちビット31～24までしか書き換え可能レジスタが実装されていないので，8Mバイト単位のアドレスを意味するFF80_0000hなどの値を書き込んでも，FF00_000hとなってしまうからです．よって，要求アドレス空間サイズが大きくなればなるほど，配置可能なアドレスの自由度もなくなってくることも意味します．

ここでたとえば，ベースアドレスレジスタ2本がそれぞれ64Kバイトと16Mバイトのメモリを要求するデバイス番号1，そしてベースアドレスレジスタが1本だけで1Mバイトのメモリを要求するデバイス番号7のデバイスがあったとします．また，ここでは4Gバイトの最上位アドレスからアドレス空間を割り当てていくことにしましょう．

単純にデバイスやベースアドレスレジスタを検索した順番にアドレスを割り当てていくとすると，まず最初のデバイス番号1のベースアドレスレジスタ0はFFFF_0000hからFFFF_FFFFhまでの64Kバイトのアドレスが割り当てられます．同じくデバイス番号1のベースアドレスレジスタ1では，ベースアドレスレジスタ0と重ならない領域となると，間をあけてFE00_0000h～FEFF_FFFFhまでの16Mバイトを割り当てるしかありません（もちろんもっと間を空けてもいいが，未使用空間を増やすのはむだ）．そしてデバイス番号7のベースアドレスレジスタ0は1Mバイトなので，間を空けずにFDF0_0000h～FDFF_FFFFhまでを割り当て可能です〔図8（a）〕．

この方法では途中に未使用の空間ができてしまいます．そこでベースアドレスレジスタを要求空間サイズ順にソートしなおし，たとえば大きな空間サイズ順に割り当てると，図8（b）の

Column 4

レイテンシタイマの設定

バスマスタデバイスのレイテンシタイマに設定する値をどうするかは，PCIデバイス初期化時に悩む点の一つです．

バスマスタデバイスの中には，データ転送に必要な最低帯域が決まっているものがあります．たとえば外部から1Mバイト/秒の一定レートでメインメモリにデータを取りこまなければならないデータ収集ボードの場合を想定します．ハードウェアの仕様として，内部のバッファサイズが1Kバイトで1ワード転送するのにPCIバスで4クロックかかるとします．1Mバイト/秒というレートは，1バイトあたり1μsの転送時間です．バッファサイズが1Kバイトなので，1024μsごとに1Kバイト以上のレートで，言い換えると10μsごとに10バイト以上でデータ転送ができれば，バッファオーバフローを発生させずにデータを転送できるわけです．

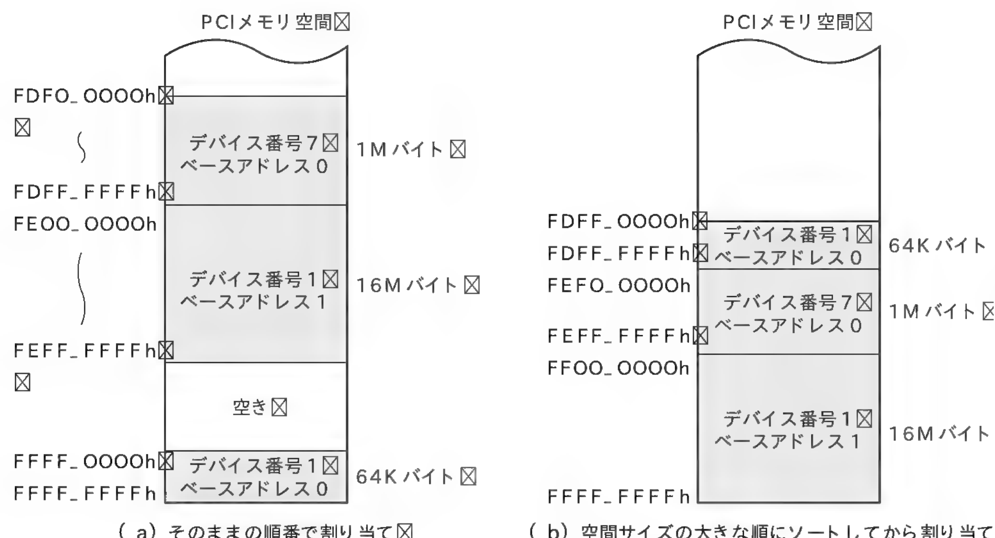
1ワード転送するのに4クロックかかるというハードウェアの仕様から，PCIバス上（32ビット幅）で10バイトを転送するには12クロックが必要です．PCIバスのクロックを33MHzで計算すると約364nsの時間になります．つまり最大10μsの間（最大レイテンシ）に，最小364nsの間（最小グラント）だけPCIバスを使わせてくれれば，転送が間に合うというわけです．これらのレジスタは実際には250ns単位の時間を格納するので，この例の場合，最大レイテンシは28h，最小グラントは端数を切り上げて2にします．

このように最大レイテンシ/最小グラントレジスタによって，そのデバイスが必要なデータ転送帯域を判定することができます．しかしバスというのはそのシステムに実装されたデバイスで共有して使う，しかも転送能力には上限がある，限られたリソースです．他にもっと広い転送帯域を要求するデバイスがいるかもしれません．

そこでPCI BIOSは，各デバイスの最大レイテンシ/最小グラントレジスタのデータを集めて，限られたPCIバスの転送帯域を，それぞれどう分配するかを考えます．その結果を保持するための，リード/ライト可能なレジスタがレイテンシタイマレジスタです．

とはいえ，具体的にどう分配するかは議論が分かれるところでしょう．筆者の作成したPCI BIOSでは，最大レイテンシ/最小グラントレジスタは参照せずに，40hまたは20hをすべてのバスマスタデバイスに対して設定しています．

〔図8〕リソース割り当てのようす



（a）そのままの順番で割り当て　（b）空間サイズの大きな順にソートしてから割り当て

〔リスト5〕リソース割り当て処理

```
    ソート処理が入る

if (IoCount > 0) {/* I/O空間を使うデバイスがいる場合 */
    /* I/Oリソースアドレス割り当て */
    for(i=0;i<IoCount;i++){
        /* 次に処理するI/O要求容量のアドレス単位がI/Oベースアドレス単位より大きい場合 */
        BaseCapa=(BaseIo[i].BaseAdr^0x0000ffff) & PCI_IoAddress;
        if (BaseCapa != 0) {        /* 半端なアドレスが発生する場合 */
            /* 半端分のアドレスを前送り */
            PCI_IoAddress=PCI_IoAddress-BaseCapa;
        }
    /* ISAバスブリッジの存在を考慮する場合の処理 */
        /* ISAバスが使用するI/O空間の場合は次のPCIバスI/O使用可能アドレスへ */
        if ((PCI_IoAddress-1) & 0x300) {
            PCI_IoAddress=((PCI_IoAddress-1) & 0xfc00)+0x100;
        }
        /* I/Oアドレス割り当て */
        PCI_IoAddress=PCI_IoAddress-((BaseIo[i].BaseAdr^0x0000ffff)+1);
        BaseIo[i].BaseAdr=PCI_IoAddress;
    }
    /* 次のバスの割り当て開始アドレスを設定 */
    PCI_IoAddress=PCI_IoAddress & 0xf000;
    /* PCIブリッジの範囲指定が4Kバイト単位なので */
}

if (MemCount > 0) {    /* 通常メモリ空間を使うデバイスがいる場合 */
    if (PCI_MemAddress != 0) {    /* 4G最上位アドレスの時はチェックしない */
        /* メモリ最大要求容量のアドレス単位が
                                    メモリベースアドレス単位より大きい場合 */
        BaseCapa=(BaseMem[0].BaseAdr^0xffffffff) & PCI_MemAddress;
        if (BaseCapa != 0) {        /* 半端なアドレスが発生する場合 */
            /* 半端分のアドレスを前送り */
            PCI_MemAddress=PCI_MemAddress-BaseCapa;
        }
    }
    /* メモリリソースアドレス割り当て */
    for(i=0;i<MemCount;i++){
        /* メモリアドレス割り当て */
        PCI_MemAddress=PCI_MemAddress-((BaseMem[i].BaseAdr^0xffffffff)+1);
        BaseMem[i].BaseAdr=PCI_MemAddress;
    }
    /* 次のバスの割り当て開始アドレスを設定 */
    PCI_MemAddress=PCI_MemAddress & 0xfff00000;
    /* PCIブリッジの範囲指定が1Mバイト単位なので */
}

if (PreCount > 0) {    /* プリフェッチメモリ空間を使うデバイスがいる場合 */

    メモリ空間と同様

}
```

ように途中に空き領域もなくきっちり詰めて空間を割り当てることができるのです（リスト5）．このように，一つのデバイスで複数のベースアドレスが要求されているとき，必ずしも隣り合うアドレスに配置する必要はありません．それぞれ離れたアドレスを指定することも可能なのです．

● リソースの設定

要求空間サイズ順にソートした順番に，各デバイスの各ベースアドレスレジスタにアドレスを割り当てていきます．ベースアドレスレジスタを設定した後に，コマンドレジスタのI/O空間イネーブル，メモリ空間イネーブル，バスマスタイネーブルの各イネーブルビットをセットしていきます（リスト6）．

また，PCI-PCIブリッジのアドレスウィンドウの設定用に，そのバスで割り当てたアドレス範囲の先頭アドレスと終了アドレスを保持しておきます．

● 割り込みラインレジスタの設定

PCIデバイスのハードウェア的には，割り込みラインレジスタは単なる値を保持するRAMとしてしか機能していません．

PC/AT互換機の場合は，PCIバス上のINTA#～INTD#をIRQ*n*にルーティングする初期化も，PCI BIOSの重要な機能の一つです．この設定により，そのデバイスのINTA#がIRQの何番にルーティングされたかを，このレジスタに書き込み保存しておきます．

組み込み機器では，プラットホームにより割り込みの実装が異なります．同じSH-4を搭載していても，CQ RISC評価キット/SH-4PCI with

Linuxでは，PCIバス上の割り込みをISAバスブリッジ内のインタラプトルータでルーティングしてからSH-4に入力します．よって割り込みラインレジスタは，PC/AT互換機と同様にルーティングしたIRQの番号を格納するために使うようです．

逆にISAバスブリッジを使わない場合は，何らかの方法でINTA#～INTD#で割り込みが発生したことをSH-4に入力する割り込み制御回路が必要になるでしょう．その制御回路の仕様によっては，割り込みのルーティング情報を保持する必要があるかもしれません．そのような場合，この割り込みラインレジスタに書き込んで保持しておくのも手です．

● PCI-PCIブリッジのウィンドウ設定

デバイス検索ルーチンから抜ける場合，そのPCIバスがバス番号0以外なら，PCI-PCIブリッジより下のバスを検索していることになるので，PCI-PCIブリッジのアドレスウィンドウの設定が必要です．

さきほどリソースを割り当てたところで，そのバス上で割り当てたアドレス範囲の先頭アドレスと終了アドレスを保持しましたが，その値をここで設定します．ただし，開始アドレスをベースアドレスレジスタに，終了アドレスをリミットアドレスレジスタにそのまま書き込むわけにはいきません．

メモリベース/リミットの場合はビット15～4までの全12ビットが指定可能範囲なので，32ビットアドレスに換算すると1Mバイト単位のアドレスで指定することになります．たとえば図8(b)の例で考えると，割り当てた領域の先頭アドレスはFDFF_0000hですが，ベースアドレスレジスタの仕様から，FDF0_0000hを先頭アドレスとするので，ベースアドレスレジスタにはFDF0hを設定します．終了アドレスはFFFF_FFFFhですが，下位20ビットをマスクしてFFF0_0000hとなるので，リミットアドレスレジスタにはFFF0hを設定することになります．

I/O空間も，ビット範囲が異なるだけで，考え方は同様です．

また，PCI-PCIブリッジの下のデバイス検索を開始する時点で暫定的に設定したサブボーディネートバス番号を，その時点でのPCI最大バス番号に書き換えておきます(リスト1Ⓑ)．

〔リスト6〕リソース設定処理

```
/* リソース割り当て */
/* I/Oベースアドレスレジスタ設定 */
for(i=0;i<IoCount;i++){
    PCIBridge_CfgLongWrite(PCIBIOS_BusNo(BaseIo[i].PCIDevAdr),
                           PCIBIOS_DevNo(BaseIo[i].PCIDevAdr),
                           PCIBIOS_FuncNo(BaseIo[i].PCIDevAdr),
                           PCIBIOS_RegNo(BaseIo[i].PCIDevAdr),
                                       BaseIo[i].BaseAdr);
         reslt=reslt|1;      /* ビット 0 I/O空間使用 */
}
/* 標準メモリベースアドレスレジスタ設定 */
for(i=0;i<MemCount;i++){
    PCIBridge_CfgLongWrite(PCIBIOS_BusNo(BaseMem[i].PCIDevAdr),
                           PCIBIOS_DevNo(BaseMem[i].PCIDevAdr),
                           PCIBIOS_FuncNo(BaseMem[i].PCIDevAdr),
                           PCIBIOS_RegNo(BaseMem[i].PCIDevAdr),
                                       BaseMem[i].BaseAdr);
    reslt=reslt|2;      /* ビット 1 メモリ空間使用 */
}
/* プリフェッチメモリベースアドレスレジスタ設定 */
for(i=0;i<PreCount;i++){
    PCIBridge_CfgLongWrite(PCIBIOS_BusNo(BasePre[i].PCIDevAdr),
                           PCIBIOS_DevNo(BasePre[i].PCIDevAdr),
                           PCIBIOS_FuncNo(BasePre[i].PCIDevAdr),
                           PCIBIOS_RegNo(BasePre[i].PCIDevAdr),
                                       BasePre[i].BaseAdr);
    reslt=reslt|4;      /* ビット 2 メモリ空間使用 */
}

/* その他初期化&イネーブルビットセット */
for(i=0;i<DevCount;i++) {
    l=DevCtrl[i]>>8;    /* デバイス番号 */
    j=DevCtrl[i]&255;   /* ファンクション番号 */

    /* 存在するPCIデバイスすべてについて共通して行う初期化処理 */
    /* レイテンシタイマ&キャッシュラインサイズはデフォルト値固定 */
    d=(PCI_LatencyTimer<<8)|PCI_CacheLineSize;
    /* レイテンシタイマ&キャッシュラインサイズ設定 */
    PCIBridge_CfgWordWrite(PCI_BusNo,l,j,0xc,d);

    /* 割り込み使用デバイスに対する処理 */
    /* インタラプトピンレジスタ */
    d=PCIBridge_CfgByteRead(PCI_BusNo,l,j,0x3d);
    if (d != 0) {  /* 割り込みを使うデバイスの場合 */
         d=PCIBIOS_INTLine(l,d,INTLineP);
         PCIBridge_CfgByteWrite(PCI_BusNo,l,j,0x3c,d);
    }    /* 割り込み未使用デバイスでは処理不要 */

    /* 最終イネーブル処理(コマンドレジスタ イネーブルビットON) */
    /* ステータスビットクリア & バスマスタ、メモリ、I/Oイネーブル */
    d=0xffff0007;
    PCIBridge_CfgLongWrite(PCI_BusNo,l,j,4,d);
}

return reslt;
/* ビット 0:I/O使用 ビット 1:メモリ使用 ビット 2:プリフェッチメモリ使用 */
```

● VGAコントローラやISAバスブリッジが存在する場合

組み込み機器でも，ISAブリッジ内を使ってISAバスを採用する場合も考えられます．PCIバスはアドレスとデータがマルチプレクスされていますが，ISAバスは通常のローカルバスと同様にアドレスバスとデータバスが独立しているので，周辺I/Oなどが接続しやすいためです．また画面表示用にVGA互換のビデオカードを使う場合もあるでしょう．

とくに，これらのデバイスがPCI-PCIブリッジの下に接続されている場合は，PCI-PCIブリッジのブリッジコントロールレジスタも適切に設定する必要があります．

4 初期化以外にPCI BIOSに必要な機能

PCI BIOSにはPCIデバイスを初期化する以外にも，次に説明するいくつかの機能を実装したほうがよいでしょう．

● ベンダID/デバイスID検索機能

PCIはプラグ＆プレイですから，どのスロットに差し込んでも動作できなければなりません．スロットが異なると，デバイス番号はもちろん，場合によってはバス番号も異なる位置に実装される場合もあります．よってドライバやアプリケーションは，バス番号やデバイス番号を決め打ちしてコンフィグレーションレジスタにアクセスすることはできません．自分が操作すべきデバイスが，どのバス番号/デバイス番号に実装されているか，デバイスを検索する必要があります．検索の方法としては，PCIデバイスを識別するときに使われる，ベンダIDとデバイスIDを目印とします．

組み込み機器の場合，基本的にPCIデバイスはオンボードに実装された状態で固定されるので，搭載するファームウェアなどはバス番号/デバイス番号を決め打ちしてアクセスしてもかまいません．しかし，そのファームウェアをほかの環境に移植しようとしたとき，その環境ではバス番号やデバイス番号が異なると，決め打ちしていた部分をすべて書き直す必要が出てきます．ソフトウェアの移植性を考えると，PCI BIOSのデバイス検索機能でデバイスを検索し，リソース情報を取得するような構造にすべきです．

● クラスコード検索機能

ベンダIDとデバイスIDだけでなく，クラスコードでPCIデバイスを検索する場合もあります．たとえばUSBホストコントローラには，UHCIやOHCI，そしてEHCIという規格化されたホスト仕様が規定されています．これにより，ベンダやデバイスが異なっても，UHCIやOHCIに準拠したホスト仕様であれば，それに対応したホストドライバが使えます．

このような何らかの規格に準拠したデバイスを検索する場合，ベンダIDやデバイスIDによる検索ではなく，クラスコードによるデバイス検索機能が有効です．

● コンフィグレーションレジスタアクセス機能

デバイスを検索して該当のデバイスのバス番号やデバイス番号がわかったら，割り当てられたベースアドレスや割り込みリソースの情報を取得するために，コンフィグレーションレジスタを読み出す必要があります．また，デバイス固有領域に独自仕様の初期化レジスタがある場合は，PCI BIOS以外のソフトウェアがコンフィグレーションレジスタに対して書き込み動作を必要とします．

もちろん，ドライバや上位アプリケーションに，コンフィグレーションレジスタを直接アクセスさせるのもよいのですが，コンフィグレーションをサイクルさせる方法は，プラットホームによって異なる場合があります．ここでもソフトウェアの移植性を考えると，ドライバやアプリケーションに直接コンフィグレーションサイクルを発生させるのではなく，PCI BIOSの機能としてコンフィグレーションレジスタへのアクセス機能を用意したほうがよいでしょう．

用意するアクセス機能としては，バイトサイズ，ワードサイズ，ダブルワードサイズの各アクセスサイズ別に，それぞれ読み出しと書き込み機能を実装したほうがよいでしょう．

なお，プラットホームによっては，コンフィグレーションレジスタへのアクセスがダブルワードでしか行えないコントローラも存在します（組み込み向けプロセッサ内蔵のPCIコントローラにときおり見られる）．図4を見てもわかるように，コンフィグレーションレジスタにはバイトやワード単位で役割が割り当てられているレジスタがあるので，ソフトウェア的にはバイト/ワード単位でアクセスしたいところです．このようなハードウェアの場合，PCI BIOSレベルで各サイズ別のアクセス機能を用意し，バイトサイズ書き込みの場合は，いったんダブルワードでコンフィグレーションレジスタを読み出し，該当バイト位置のみ論理演算などで書き換え，それをダブルワードサイズで書き戻すようにします．

ただし，ステータスレジスタに相当するアドレスラインの書き換えには注意が必要です．ステータスレジスタは，ビットをクリアするために該当ビットに'1'を立てた値を書き込みます．よって，隣のコマンドレジスタを書き換えるために上記操作でダブルワードで書き戻したとき，もしステータスレジスタにエラービットが立っていた場合，そのビットはクリアされてしまうことになります．

やはりハードウェア的にもバイト/ワード単位でのアクセスが可能なコントローラを採用するのが正しいでしょう．

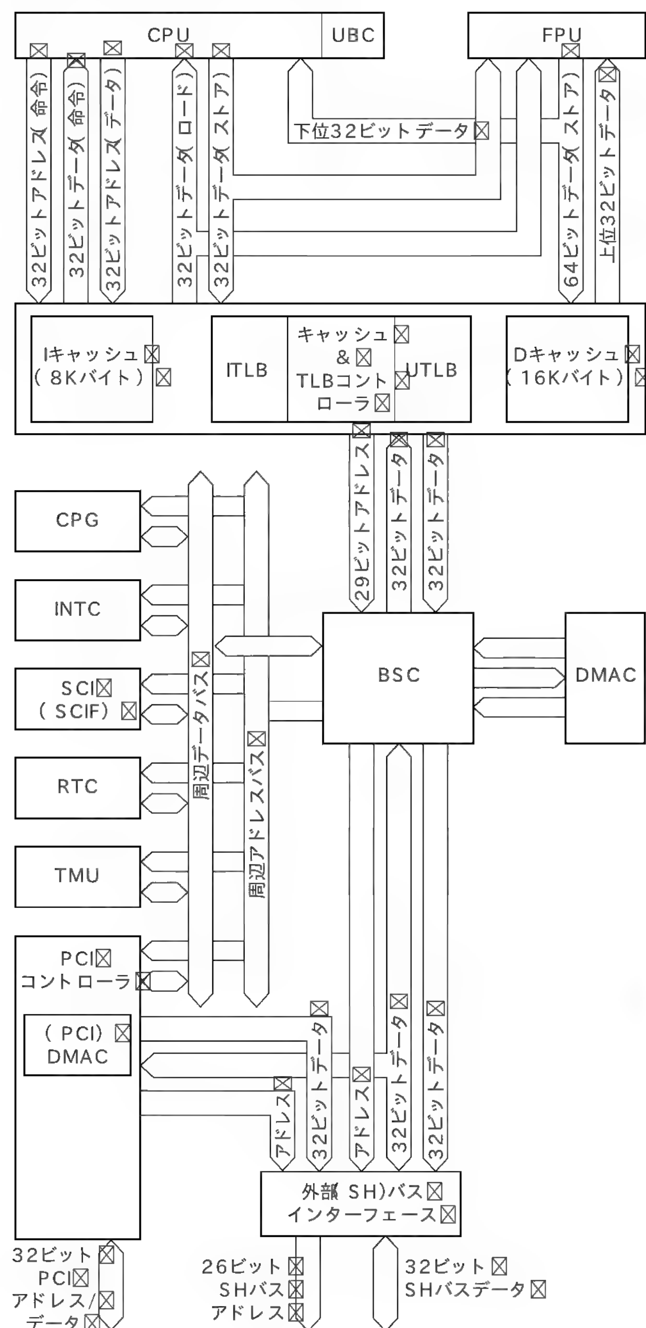
参考文献
1)『PCI LocalBus Specification Rev2.1 and Rev2.3』，PCI-SIG.
2)『PC Card Standard Release8』，PCMCIA/JEITA.
3)『PCIデバイス設計入門』，TECHI Vol.3，CQ出版（株）.

やまたけ・いちろう　来栖川電工有限会社

組み込み機器における PCIバスの実装方法

山武 一朗

第5章では，PC/AT互換機でのCPUのアドレス空間とPCI空間の関係を説明しました．最近では，組み込み向けプロセッサでもPCIコントローラを内蔵したものが増えています．組み込み向けプロセッサでは，PCI空間はどのようにマッピングされているのでしょうか．ここではSHシリーズとMIPSのV_R4100シリーズ，そしてPowerPCシリーズについて見てみましょう．



〔図A〕SH7751(SH-4)のブロック図

● SHシリーズの場合

SH-3およびSH-4では，CPUが扱えるメモリ空間は4Gバイトあるのですが，そのうち上位3ビットは，メモリ保護/MMUアドレス変換領域などの区別のために使われています．よって実際のアドレス空間としては512Mバイトしかありません．さらにそのうちの64Mバイトは CPU内蔵機能の制御レジスタがマッピングされていて，実際に使える外部アドレスは，7本あるチップセレクトを入れて448Mバイトとなります．

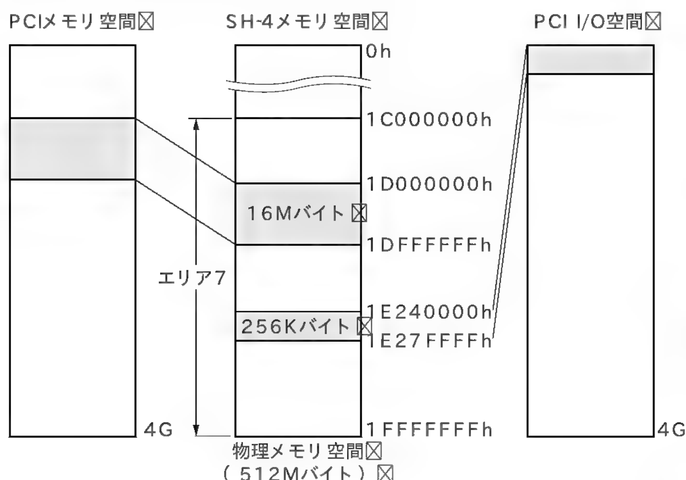
さて，SH-4シリーズの中でPCIバスコントローラを内蔵したものにSH7751があります(図A)．SH7751では，CPUが直接PCIメモリ空間をアクセスしようとした場合，16MバイトのPCIメモリウィンドウが1面しかありません(図B)．つまり，リニアにアクセスできるのは4Gバイト中16Mバイトしかないことになります．

同じSHシリーズでもSH-3では，SH7708で外部メモリ空間だったエリア1を，SH7709ではCPU内蔵機能領域に割り当てるなど，外部メモリ空間を削って機能を実装した品種もあります．SH7751ではSH7750との互換性を重視したためか，PCIメモリ空間ウィンドウがエリア7の空間に押し込まれているのです．

● V_R4100シリーズの場合

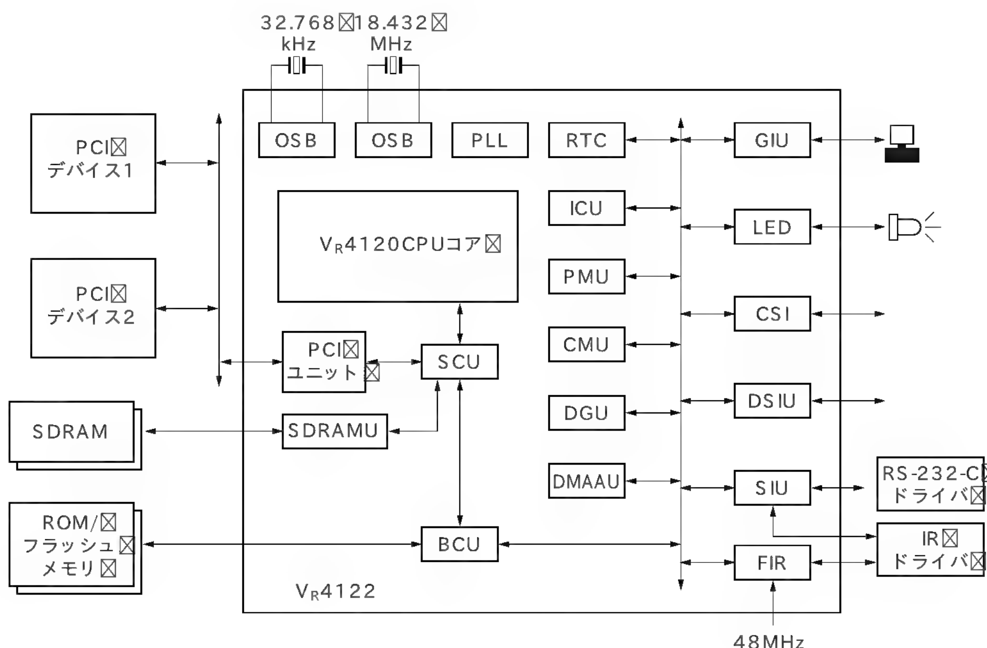
V_R4100シリーズでPCIコントローラを内蔵したものには，V_R4122/4131/4133があります．いずれもほぼ同じ仕様のPCIコントローラが内蔵されています．図CにV_R4122のブロック図を示します．

MIPSプロセッサの物理アドレス空間は，32ビットモードの場合，00000000h～1FFFFFFFhまでの512Mバイトとなります．この空間をROMやRAM，内蔵レジスタ空間やI/O空間，PCI空間に分け

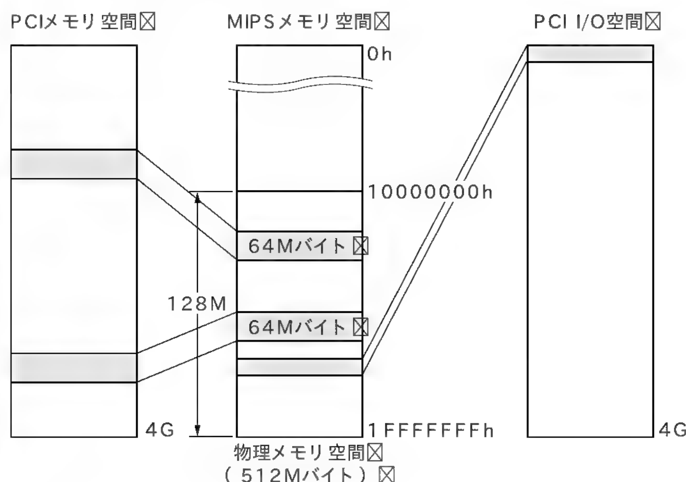


〔図B〕SH7751(SH-4)のPCIメモリマップ

〔図C〕V_R4122のブロック図



〔図D〕V_R4122のPCIメモリマップ



て使います．このうちPCI空間は，10000000h～17FFFFFFhまでの128Mバイトに割り当て可能です．V_R4000シリーズでは，SHのようにPCIへのアクセスウィンドウが固定ではなく，この範囲でPCIメモリウィンドウを二つ，PCI I/Oウィンドウを一つ開くことができます．またウィンドウ1本あたり最大64Mバイトのサイズまでとなります(図D)．

PC/AT互換機の場合，CPUのメモリ空間である4Gバイトがそのまま外部アドレスとしてまるまる使えるわけですが，ほとんどの組み込み向けプロセッサの場合，このようにただでさえ狭い物理アドレス空間内に，さらにウィンドウを仕切ってPCI空間を確保しています．

● PowerPCシリーズの場合

SHやV_R4100シリーズでは非常に狭いPCI空間でしたが，同じ組み込み系プロセッサでも，PowerPCは若干ようすが異なります．ここではMPC8245(モトローラ)について見てみましょう．

PowerPCではプロセッサアーキテクチャとして，4Gバイトのメモリ空間のうち前半2Gバイトをローカルメモリ空間に，後半2Gバイト－32MバイトをPCIメモリ空間に，残りの32MバイトをPCI I/O空間とコンフィグレーション空間などに割り当てています(図E)．これは，組み込み向けプロセッサMPC8245でも例外ではありません．PCIメモリ空間は2Gバイト弱という広大な空間を，CPUコアからリニアにアクセスできることになります．

● バスマスタ転送の場合

上で説明したPCIメモリウィンドウは，あくまでCPUコアがPCIメモリ空間にアクセスしにいく場合に使う空間です．CPUがPCIメモリを読み書きする，いわゆるPIO転送で必要となる空間ということです．

すでに説明しているように，PCIバスを採用して性能を出すには，バースト転送が必須です．バースト転送でデータ転送を行うには，バスマスタデバイスを使ってバスマスタ転送を行うのが，正しいPCIの使い方といえるでしょう．

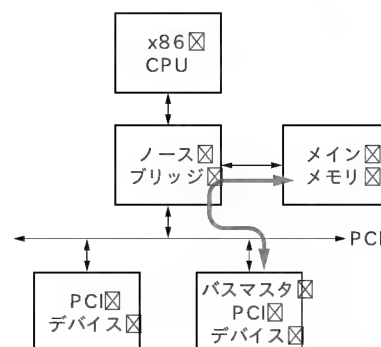
バスマスタデバイスの一般的な使い方としては，ホストCPUのメインメモリ上にデータ転送バッファを用意して，その領域をPCIバスマスタデバイスにアクセスさせます．図FはPC/AT互換機での場合ですが，組み込み向けプロセッサでは図Gのように，内蔵PCIコントローラがバスマスタからのアクセスを受け，それを内蔵のSDRAMコントローラ経由でローカルRAMであるSDRAMにアクセスする形になります．一般的に，この転送ではPIO転送用の空間は使いません．PCIコントローラを内蔵した組み込み向けプロセッサでは，ほとんどの場合，ローカルメモリ空間をPCIメモリ空間の任意のアドレスにマッピングする機能をもっています．これにより，PCIバス上のバスマスタが組み込みプロセッサの制御しているローカルRAMへ，直接バスマスタ転送できるのです．

このように，すべてのデータ転送をバスマスタに行わせるのであれば，CPUコアがPCI空間にアクセスする必要はありません．しかし

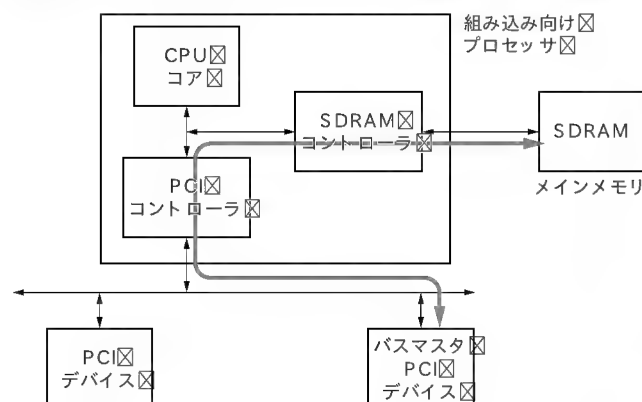
〔図E〕PowerPCのメモリマップ



〔図F〕
PC/AT互換機におけるバスマスタ転送のデータの流れ



〔図G〕
組み込み向けプロセッサにおけるバスマスタ転送のデータの流れ



バスマスタデバイスといえど，ホストCPUからの指示なしに勝手に動き回れるわけではありません．バスマスタ転送を制御するための制御レジスタをメモリ空間かI/O空間に実装し，ホストCPUからそのレジスタを設定してもらうことで動くことができます．よって，バスマスタ制御レジスタはCPUコアから見えるところに配置しなければなりません．

● 組み込み機器におけるPCIバスの使い方

たとえば，PIO転送とバスマスタ転送の両方に対応したPCIデバイスがあるとします．2本のベースアドレスレジスタを実装し，片方のベースアドレスの領域にはPIO転送のためのデータバッファが数Mバイト，もう片方のベースアドレスレジスタの領域にはバスマスタ転送のための制御レジスタが数十Kバイトマッピングされています．

このデバイスをバスマスタ転送だけで使うのであれば，数Mバイトのデータバッファ空間をCPUコアに見せる必要はありません．データ転送そのものはバスマスタがやってくれるので，CPUコアにはバスマスタ転送を制御する，数十Kバイトのバスマスタ制御レジスタ空間を見せればよいわけです．

● フレームバッファの類はターゲットデバイスが多い

しかし，PCIデバイスによってはバスマスタ機能は実装されておらず，それでいて広大なメモリ空間を必要とするものがあります．筆者の会社で過去に開発したものでは，背景にビデオ映像を写しながら手前にスーパーインポーズ風に文字をスクロールで流し，それを合成してRGB出力するというビデオカードがありました．

このカードのスーパーインポーズ映像領域は，フォアグラウンドカラーと文字の背景にあたる透明色を指定するといったような単純なものではなく，フルカラーCGをαチャネル付きでビデオ映像と合成するもので，フルカラーフレームバッファビデオカードそのものの機能をもちます．しかも解像度がVGAなどという低解像度ではなく，XGA以上(最終的にはSXGAになった)が要求され，さらにスクロールバッファなどのため，表示解像度の数面分を保持するという仕様です．フルカラー+αチャネルなので1ピクセル32ビットとすると，16M～64Mバイトものフレームバッファ空間が必要になります．

このビデオカードは，フレームバッファ内の矩形領域転送機能などのアクセラレーション機能は実装していますが，ホストCPUのメインメモリから表示データをバスマスタで自分で読みにいく機能までは実装していません．

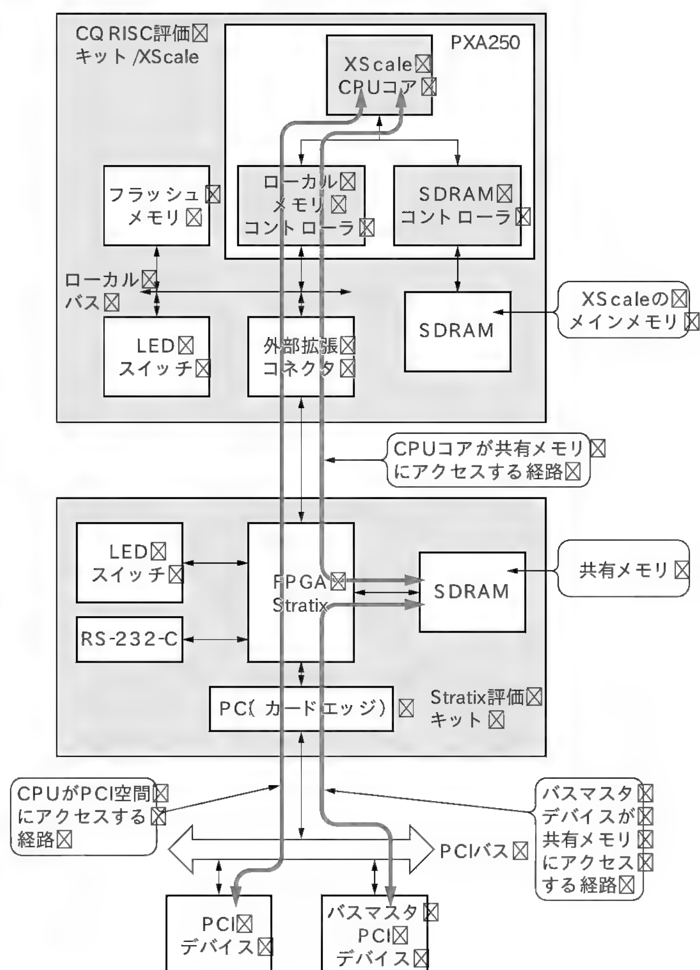
このようなPCIデバイスの場合，ホストCPUがフレームバッファ領域をリニアにアクセスできないと，描画プログラム中に複雑なアドレス変換やバンク切り替え処理などが必要になってしまいます．

● PIO転送用空間が必要な場合は外付けPCIブリッジで

筆者の会社では，PCIコントローラを内蔵していない組み込み向けプロセッサに，FPGAを外付けしてPCIへブリッジするというシステムを何度か設計しています．

SH-4の場合，エリア1から6までメモリや外付けI/O機能をフルに実装することはまずありません．筆者のこれまでの経験では，エリア0にはフラッシュメモリを数十Mバイト実装し，RAMが64Mバイトでは足りないという場合でもエリア2と3をSDRAMとして

〔図H〕バスマスタ対応共有メモリを実装した例



最大128Mバイトまで実装できます．またCPU内蔵機能やPCIバス上に実装できない外付けのI/O機能などをエリア1に実装すれば，残りエリア4～6をすべてPCI空間に割り当てることができます．

次号で掲載予定のXScale徹底活用連載記事では，CQ RISC評価キット/XScaleとStratix評価キットをつなげて，PXA250をPCIバスにブリッジするPCIコントローラをStratixで設計しています．この場合，PXA250のスタティックエリア3～5をPCI空間として割り当てました．

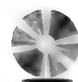
● バスマスタPCIデバイスへの対応

CPUの外部バスにFPGAなど外付けでPCIコントローラを実現する方法は，CPUコアから見たPCIメモリ空間を，大容量でリニアに確保できるという利点があります．しかしこの構造では，基本的にPCIバス上のほかのバスマスタデバイスが，ローカルRAMにアクセスできません．ほとんどの組み込み向けプロセッサはSDRAMコントローラを内蔵し，これによりローカルRAMを制御しています．しかしこのSDRAMコントローラが，一般的に外部のバスマスタからアクセスされることを考慮していません．

そこで，外付けでPCIコントローラを実現する場合は，バスマスタ転送などを考慮してCPUとバスマスタでメモリを共有できる，バッファメモリを実装する方法があります(図H)．

たとえば，XScale評価キットとStratix評価キットを接続した例では，Stratix評価ボード上に実装されているSDRAMを，PCIバス側からもPXA250からもアクセスできる共有メモリ領域として使えるように設計しました．これによりPCIバスマスタは，直接PXA250が制御しているSDRAMにはアクセスできないものの，Stratix評価ボード上のSDRAMに対してバスマスタ転送を行い，データ転送終了後にPXA250がそのSDRAMにアクセスし，必要なデータを受け取るというシステムを構築できます．

やまたけ・いちろう　来栖川電工有限会社



システム起動用リソース/BIOS拡張手法

PCI拡張ROM プログラムの開発

菅原 尚伸

通常のPCIデバイスは，OS起動後にデバイスドライバを読み込んで初めて動作を始める．しかしシステム起動時に必要なデバイスの場合は，OS起動の前に何らかの初期化プログラムを走らせるしくみが必要になる．PCIボードに拡張ROMを搭載し，BIOSの初期化時に拡張ROMプログラムを実行することで，OSの起動前にPCIデバイスを初期化できる．本章では，PCI拡張ROMの起動シーケンスや，拡張ROMの構造，そして拡張ROMのプログラム開発方法として具体的に動作するサンプルプログラムを開発する．（編集部）

1 PCI拡張ROMとPCIデバイス

● PCI拡張ROMとは

PCI拡張ROMは，大きく分けると2種類のプログラムを格納するために使います．

▶ 初期化プログラム

OS起動の前から使用する必要のあるデバイスは，OSが起動してから初期化やデバイスドライバのロードをしていたのでは間に合いません．このような場合，システムの起動に必要なデバイスのための初期化プログラムを，ROMに格納しておくことができます．

▶ 機能拡張プログラム

一般的にはOSが起動してからデバイスドライバという形でプログラムをロードして使いますが，外部からプログラムを読み込んで使うのではなく，ROMの中に制御用プログラムを格納しておき，これを呼び出すことでPCIデバイスを制御するシステムも実現できます．

この方式は，システムBIOSの機能をBIOSレベルで拡張する場合によく使われるので，拡張BIOSとも呼ばれます．

● PCIバスの汎用性

PCIバスはPC/AT互換機のためのだけのバスシステムではありません．MacintoshやNEC PC-98シリーズでも使われている汎用バスです．それぞれのプラットホームで，PCI拡張ROMを必要とする場合が考えられます．このとき，拡張ROMの中に1種類のプラットホームの拡張プログラムしか格納できないのでは，そのプラットホーム専用のPCIボードとなってしまいます．

そこで，同じ1枚のボードを多数のプラットホームで共通で使えるように，PCI拡張ROMの中はそれぞれのプラットホームに対応したプログラムを複数実装できる構造になっています．

● PCIターゲットデバイスの仕様

PCI拡張ROMを実装するには，PCI拡張ROMに対応したPCIデバイスが必要です．ここではPCI拡張ROMの中身，つまりソフトウェア面について解説するので，PCI拡張ROM対応ターゲットデバイスのハードウェア設計については，コラム3（p.125）を参照してください．

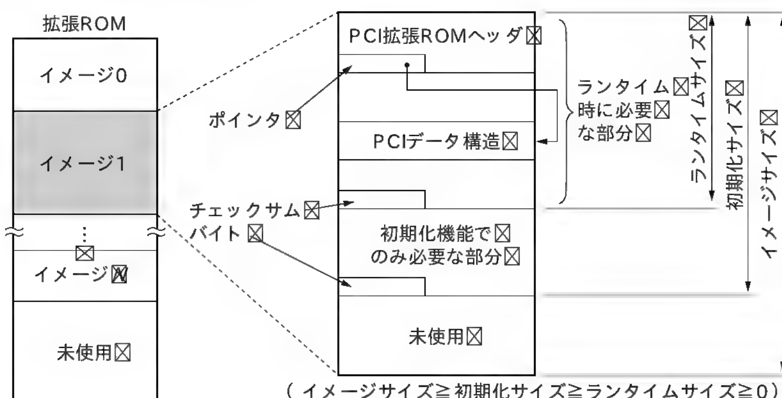
以降の説明では，ハードウェアについては完成しているという前提で解説します．

2 PCI拡張ROMとヘッダの構造

● PCI拡張ROMの構造

図1にPCI拡張ROMの構造を示します．PCI拡張ROM内は，イメージと呼ばれるブロック単位で構成されています．ROMの先頭には必ず最初のイメージが格納されています．イメージの中には後述する二つのヘッダが存在します．まずイ

〔図1〕PCI拡張ROMの構造



メージの先頭にあるのが PCI 拡張 ROM ヘッダです．もう一つは PCI データ構造と呼ばれるヘッダです．PCI 拡張 ROM ヘッダの中には PCI データ構造へのポインタがあり，その値をイメージの先頭からのオフセットとした位置から PCI データ構造が格納されます．したがって，この二つのヘッダは連続している必要はありませんが，PCI データ構造もイメージ内の先頭付近に置くのが一般的でしょう．

● イメージ/初期化/ランタイム

じつは PC/AT 互換機では，拡張 BIOS はそのまま ROM 上で実行されるのではなく，シャドウ RAM にコピーして実行されます．このしくみは PCI 拡張 ROM でも同様です．また，拡張 BIOS を格納するシャドウ RAM 領域は約 128K バイト程度と，非常に狭いのです．

さて，プログラムには，初期化時にのみ必要なコードやデータ，そして起動した後にも DOS プログラムや場合によっては OS からコールされるコード（ランタイム）部分などがあります．初期化時にのみ実行するプログラムを，システム稼働状態でもメモリ内に格納しておくのはメモリのむだ使いといえます．そこで拡張 ROM 内を初期化時に必要な部分とランタイムとに分け，システム起動時に初期化部を実行したあとは，初期化部を削除したサイズだけメモリを確保してランタイムのみを格納することで，メモリの占有率を下げることができます．

システム稼働時に必要なプログラムやデータを格納したランタイム部分と，それに初期化時に必要なコードを含めた初期化部分，そして全体をまとめてイメージと呼びます．イメージといった場合は初期化部分も含みます．また初期化部分にはランタイムも含みます．

一般的には初期化サイズ＝イメージサイズとなるでしょう．初期化部分にも含めない，たとえばプログラムにはまったく関係のない，ROM の内容を人間が管理するときに見やすいように，ROM の名前やバージョン名などを ASCII コードで埋め込むなどのような場合，初期化部分の外に配置すれば初期化サイズ＜イメージサイズとなるでしょう．

PCI 拡張 ROM は，このイメージ単位で一つ，または複数をまとめて格納します．なおランタイムサイズや初期化サイズ，イメージサイズは 512 バイト単位で確保します．たとえば必要な初期化プログラムサイズが 1025 バイトになった場合には，初期化サイズを 1536 バイトにします．

● PCI 拡張 ROM ヘッダ

表 1 に PCI 拡張 ROM ヘッダの構造を示します．PC/AT 互換機では表 1（a）のようなフォーマットで使われています．オフセット 2 に格納されている初期化サイズは，512 バイト単位の数値なので，たとえば初期化サイズが 8K バイトのときは 10h を格納します．

オフセット 3 にはジャンプ命令を置きます．システム BIOS がこの PCI 拡張 ROM を呼び出すときは，ここに FAR CALL をしてきます．よって，ここからさらに実際に初期化プログラムが格納されているアドレスに，もう一度ジャンプするわけです．

オフセット 6 からの 18 バイトは，デバイス固有に使用できます．社名や PCI デバイス/ボードの製品名，ROM のバージョンなどを ASCII コードなどで記録してもかまいません．

オフセット 18h に格納される PCI データ構造へのポインタは，PCI データ構造が格納されるイメージ領域の先頭からのオフセット値を入れます．

● PCI データ構造

表 2 に PCI データ構造を示します．PCI データ構造にはリビジョンがあり，現在はリビジョン 0 が定義されています．

〔表 1〕PCI 拡張 ROM ヘッダの構造

オフセット	長さ	説　明
00h	1	ROM シグネチャ　55h
01h	1	ROM シグネチャ　AAh
02h ～ 17h	22	システムアーキテクチャ，CPU アーキテクチャにより個別定義される
18h ～ 19h	2	PCI データ構造へのポインタ

（a）一般の定義

オフセット	長さ	説　明
00h	1	ROM シグネチャ　55h
01h	1	ROM シグネチャ　AAh
02h	1	初期化サイズ　単位は 512 バイト
03h	3	BIOS はこのアドレスに FAR CALL を行う．ここにはジャンプ命令を置く
06h ～ 17h	18	デバイス固有に使用可能
18h ～ 19h	2	PCI データ構造へのポインタ

（b）PC/AT 互換機での定義

〔表 2〕PCI データ構造（リビジョン 0 の定義）

オフセット	長さ	説　明
00h ～ 03h	4	PCI データ構造シグネチャ“PCIR”の文字列を格納．P が先頭
04h ～ 05h	2	ベンダ ID
06h ～ 07h	2	デバイス ID
08h ～ 09h	2	バイタル製品データ（VPD）へのポインタ．未使用の場合は 0
0Ah ～ 0Bh	2	PCI データ構造の長さ　単位はバイト
0Ch	1	PCI データ構造のリビジョン
0Dh ～ 0Fh	3	クラスコード
10h ～ 11h	2	イメージサイズ　単位は 512 バイト
12h ～ 13h	2	拡張 ROM のリビジョン
14h	1	コードタイプ 00h：x86 CPU ＆ PC/AT 互換機 01h：PCI のためのオープンファームウェア標準 02h ～ FFh：予約
15h	1	インジケータ ビット 7 “1”：この次にもイメージあり “0”：これで最後のイメージ
16h ～ 17h	2	予約

注：各種 ID やポインタなど複数バイト数からなる場合は，低位アドレスに下位側のバイトが格納される（リトルエンディアン）．

PCIデータ構造シグネチャは"PCIR"という文字列がASCIIコードで格納されています．ベンダIDやデバイスIDは，PCIデバイスのベンダIDとデバイスIDと同じ値を入れます．

PCIデータ構造の長さはリビジョン0では24バイト(18h)となります．格納サイズが2バイトなので18h，00hと格納します．PCIデータ構造のリビジョンには当然0を格納します．

オフセット0Dhからのクラスコードには，PCIデバイスのクラスコードを格納します．基本クラス，サブクラス，プログラムI/Fの順番です．オフセット10hにはイメージサイズを格納しますが，こちらは格納サイズが2バイトとなっています．たとえばイメージサイズが4096バイトなら08hとなるので，08h，00hと格納します．

拡張ROMのリビジョンには任意の値を格納してかまいません．格納サイズが2バイトあるので，1バイト目をメジャーバージョン，2バイト目をマイナバージョンとして使うなど，使い方は設計者に任されています．

コードタイプは表2にある値のいずれかを格納します．本書ではPC/AT互換機を想定しているので，後述するサンプルプログラムでは00hを入れています．

インジケータは，複数のイメージがある場合でこのイメージの後にさらにイメージが続いているときビット7を"1"にします．イメージが一つしかない場合，もしくは複数のイメージの最後に配置されているイメージの場合は，ビット7を"0"とします．他のビットは将来の拡張用となっているので，現状ではビットを立てずにクリアしておきます．

● チェックサム

PCI拡張ROMではもう一つ注意点として，イメージ単位でチェックサムが必要です．イメージ領域の先頭から最後までをバイト単位で読み出してすべて加算します．桁溢れは無視してかまいません．その結果が0となるようにデータを調整します．

● PC/AT互換機での動作

チェックサムエラーはもちろん，ベンダIDやデバイスID，クラスコードの値が拡張ROM内のヘッダの記述と実際のPCIデバイスとで異なると，システムは何らかのエラーが発生したとみなして，PCI拡張ROMを実行しないことがあります．この場合，BIOSのメーカーによっては，何事もなかったかのようにPCI拡張ROMがないときと同じように起動するものもあれば，警告の意味でビープ音を発生したり，起動途中の画面に警告メッセージを表示するものもあります．逆に某メーカー製のBIOSでは，ベンダIDとデバイスIDさえ一致すれば，クラスコードはおろかチェックサムさえ一致していないのに，PCI拡張ROMを実行するものもあります(なんていい加減な……)．

3 PCI拡張ROMプログラムの起動シーケンス

次に，PCI拡張ROMプログラムがどのような手順を経て起動するのか，そのシーケンスについて説明します．以降ではBIOSがどのようにしてPCI拡張ROMを呼び出すかを，BIOSの立場に立って説明します．ここで説明しているシーケンスは，一般的な処理手順について説明しているので，BIOSの種類によっては，詳細が異なる手順で実行されている場合もあります．

● 拡張ROMベースアドレスレジスタの有無確認

PCI拡張ROMがあろうとなかろうと，PCIデバイスの初期化処理(POST：パワーONセルフテスト)は同じです．BIOSはPOST時に，各PCIデバイスが必要としているベースアドレスや割り込みを適切に割り当てていきます．

BIOSはひととおりPCIデバイスのコンフィグレーションを終えた後に，PCI拡張ROMの検索を始めるようです．まず，PCIコンフィグレーションレジスタの中の拡張ROMベースアドレスレジスタ(レジスタアドレス30h)に，実際にレジスタが存在しているかどうかを確認します．確認はベースアドレスレジスタと同様にFFFF_FFFEhを書き込んだ後，このレジスタを読み出して値が0以外になっているかどうかで判断します．ここに書き換え可能なレジスタが実装されていれば，そのPCIデバイスはPCI拡張ROMを実装可能なデバイスであると判断できます．書き換えできなければ，その時点でPCI拡張ROMも存在しえないことがわかるので，これ以降の拡張ROMに関する処理は行われません．

● PCI拡張ROMのマッピング

PCI拡張ROMを実装可能なデバイスであると判断できた場合は，このレジスタに適切なアドレスを書き込み，最下位ビットに"1"を立て，さらにPCIコンフィグレーションレジスタのコマンドレジスタ(レジスタアドレス4)の中のメモリイネーブルビットをセットして，PCIのメモリ空間にPCI拡張ROMをマッピングします．

そのPCIデバイスがPCI拡張ROMを実装可能であるとして，必ずしもPCI拡張ROMが存在するとは限りません．PCIデバイスにはその機能があっても，実際にROMを実装していなければプログラムは存在しないからです．そこで，PCIメモリ空間にPCI拡張ROMをマッピングしたあと，本当にそこにPCI拡張ROMが存在するか，ヘッダを読み出して確認するわけです．

● イメージごとのヘッダ解析

まず拡張ROMの先頭にROMシグネチャである55h AAhが書かれているかどうかを読み出します．これが読み出せなければ，その時点でPCI拡張ROMが存在しないと判定されます．次にPCIデータ構造へのポインタが読み出され，それにしたがってPCIデータ構造のPCIデータ構造シグネチャである"PCIR"という文字列を確認します．ROMシグネチャがあるにもかかわらず，PCIデータ構造シグネチャがない場合は，最初に読み取れた55hとAAhがたまたまそのような値だったとして，やはりPCI拡張ROMは存在しないと判断します．

ここまでくるとBIOSは，PCI拡張ROMは存在していると判断します．PCI拡張ROM内は複数のイメージが存在する可

能性もあるので，それぞれのイメージごとに，その内容が正しいかどうかを確認します．

まずベンダIDやデバイスID，クラスコードをチェックして，PCIデバイスと一致しているかを確認します．またPC/AT互換機の場合はコードタイプが00hであるかどうかも確認します．そしてPCIデータ構造のオフセット10hに格納されているイメージサイズを読みとり，PCI拡張ROM内のイメージ領域をそのサイズ分だけシャドウRAM上に転送します．転送後チェックサムを確認し，チェックサムが合っていれば，転送したPCI拡張ROMのイメージの内容は正しいと判断されます．

ここで確認したイメージが該当PCIデバイスの各種IDと一致しない場合は，次のイメージを確認します．そのために今確認したイメージのインジケータ(PCIデータ構造のオフセット15h)を読み取り，ビットが立っていれば，このイメージの後に次のイメージが存在することがわかります．今確認したイメージのサイズはPCIデータ構造内に記述されているので，その値から次のイメージが今確認しているイメージの先頭から何バイト目以降に格納されているかがわかります．こうして次のイメージにROMシグネチャ(55h AAh)があるかどうかを確認し，以下同様にヘッダを解析していきます．

インジケータのビットが立っていなければ，これ以上イメージ領域はないと判断して，PCI拡張ROM内の検索を終了します．

有効なイメージがあってもなくても，PCI拡張ROM内を最後のイメージまで確認し終えたら，PCIコンフィグレーションレジスタの拡張ROMベースアドレスをゼロクリアし，PCIメモリ空間からPCI拡張ROMを切り離します．

● 拡張ROMプログラムの起動

さて，内容が正しいと判断されたイメージがある場合は，転送したシャドウRAMの先頭からオフセット3のアドレスをFAR CALLします．ちなみにこのとき，引き数としてAXレジスタに以下のパラメータが格納されています．

ビット15～8：バス番号
ビット7～3：デバイス番号
ビット2～0：ファンクション番号

これらのパラメータにより，自分が制御すべきPCIデバイスが，PCIバスのどこに実装されているかがすぐにわかります．しかもこのビット割り当ては，PCI BIOSコールですぐに使えるPCIアドレスの形式になっています．

PCI拡張ROMが呼ばれる段階ではOSやファイルシステムはロードされていません．当然ながらファイルの読み書きはできません．またDOSのファンクションコールも使えないので，BIOSコールによりプログラムを制御します．また，PCI拡張ROMはシャドウRAMに転送してから実行されますが，その転送アドレスがどのシステムでも同じとは限りません．絶対アドレスによるジャンプ命令は使えません．

レジスタの使用上の注意点としては，スタックポインタ(SP)やスタックセグメント(SS)，コードセグメント(CS)は，コール時の値そのままで使います．一般的にPCI拡張ROMは64Kバイト以内に作るので，煩雑なセグメント切り替えは必要ないでしょう．また使用可能なスタックの制限もとくにありません．一般的なプログラミングの範囲内なら，スタックは問題なく使えます．

● 拡張ROMプログラムの終了

すでに説明したようにPCI拡張ROMプログラムには，初期化部とランタイム部とに分けられます．シャドウRAM上でのメモリ占有バイト数を少しでも減らすため，ランタイム時には不要な初期化ルーチンを，初期化プログラム実行後に切り離してから，制御をBIOSに戻します．シャドウRAM上でPCI拡張ROMプログラムの初期化プログラムを実行中は，この領域は書き換え可能になっています．

切り離すといっても特別な処理は必要ありません．シャドウRAM上に転送されたイメージのPCI拡張ROMヘッダの中の初期化サイズと，PCIデータ構造内にあるイメージサイズの値を書き換えて，サイズを小さくするだけです．もちろん，このときチェックサムが正しくなるようにチェックサムバイトも書き換えることに注意してください．

BIOSに戻るときは`RET`命令を実行するだけです．この後BIOSはこのシャドウRAM領域を，書き込み不可能な設定にして，ROMと同様の状態にします．

なお，そのシステムで有効なイメージがPCI拡張ROM内に複数あった場合は，少なくとも筆者がテストしたマシンでは，最初に発見した有効なイメージしか実行しませんでした．というより，最初に有効なイメージを発見した場合は，その後にさらに別のイメージがあっても検索しないということのようです．

4 PCI拡張ROM開発支援ツール

● PCI拡張ROM開発

PCI拡張ROMプログラムの開発には，アセンブラが必要です．ここではMASMの使用を想定しています．

● ヘッダ生成/イメージサイズ計算/チェックサム

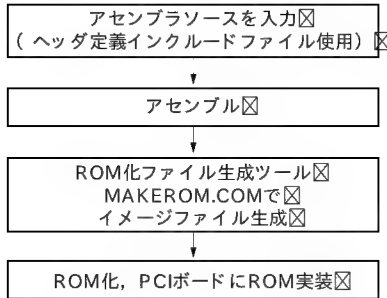
以上説明してきたように，PCI拡張ROMは各種ヘッダやイメージサイズの計算，そしてチェックサムの計算などをきっちり合わせないと，正しいPCI拡張ROMになりません．これをいちいち手作業でやっていたのでは効率があがりません．

そこで，PCI拡張ROMプログラムの開発を支援するための，アセンブラソース用のインクルードファイルや，ROM化ファイル生成ツールを作成しました．図2にこれらツールを使った場合のPCI拡張ROMの開発手順を示します．

● アセンブラソース用のインクルードファイル OPTROM.INC

PCI拡張ROMヘッダとPCIデータ構造を定義したインク

〔図2〕PCI拡張ROM開発支援ツールを使ったプログラム開発手順



〔リスト1〕ヘッダ定義インクルードファイルOPTROM.INC

```
;******************************************
;PCIR定義
;******************************************
PCIRS STRUC

PCIR_SIGN      DB 'PCIR'
PCIR_VENDOR_ID DW  ?
PCIR_DEVICE_ID DW  ?
PCIR_VPD       DW  ?
PCIR_LEN       DW  SIZE PCIRS
PCIR_REV       DB  00H
PCIR_CLASS_PROG DB  ?
PCIR_CLASS_SUB  DB  ?
PCIR_CLASS_BASE DB  ?
PCIR_IMAGE_LEN DW  ?
PCIR_REV_LVL   DW  ?
PCIR_CODE      DB  ?
PCIR_ID        DB  ?
PCIR_RSV       DB  2 DUP(?)

PCIRS ENDS

;******************************************
;ROM Header Macro定義
;******************************************
ROM_HEADER MACRO

        ORG     0
;---------------------------------------
;PCI拡張ROMヘッダ構造定義
;---------------------------------------
ROM_SIGN    DW   0AA55H                   ;PCI拡張ROMヘッダ(固定)
ROM_SIZE    DB   0                        ;初期化サイズ(MAKEROM.COMが設定する)
ROM_JMP     LABEL FAR
            JMP  NEAR PTR &initCodeLabel  ;拡張ROMプログラムへのジャンプ
ROM_DATA    DB   strROMData               ;デバイス固有領域(最大18バイト)
ROM_DATA_END LABEL BYTE
            ORG  18h
ROM_POINT   DW   PCIR                     ;PCIデータ構造へのポインタ(自動計算)
            .ERRNZ ROM_POINT LT ROM_DATA_END ;ROM_DATAが18バイトを超えた場合エラーとなる

;---------------------------------------
;PCIボード名/拡張ROMバージョンなど
;---------------------------------------
            DB   strROMName
;---------------------------------------
;PCIデータ構造定義
;---------------------------------------
             ALIGN 4
PCIR        LABEL PCIRS
            PCIRS<,pcirVendorID,pcirDeviceID,pcirVPD,,,¥
                   pcirClassBase,pcirClassSub,pcirClassProg,,¥
                   pcirRevLvl,pcirCode,pcirID>

          ENDM

;********************************************
; ROM_HEADERマクロ用デフォルト値
;********************************************
strROMData     CATSTR <'PCI Option ROM'>      ;デバイス固有領域(最大18バイト)
strROMName     CATSTR <'Option ROM Image',0> ;PCIボード名/拡張ROMバージョンなど
pcirVendorID  = 0000h
pcirDeviceID  = 0000h
pcirVPD       = 0000h
pcirClassProg = 00h
pcirClassSub  = 00h
pcirClassBase = 00h
pcirRevLvl    = 0000h
pcirCode      = 00h
pcirID        = 00h
initCodeLabel CATSTR <INIT_CODE>
```

ルードファイルです(リスト1)．これを，ユーザーが開発するPCI拡張ROMのアセンブラプログラムにインクルードして使います．ユーザーのアセンブラプログラムでは，pcirVendorIDやpcirClassBaseなど，ユーザーが定義するパラメータを代入すれば，アセンブル時に自動的にヘッダを生成してくれます．

なお，注意点として，PCIデータ構造中のバイタル製品データ(pcirVPD)は0でかまいません．また，コードタイプ(pcirCode)はx86&PC/AT互換機用のコードなので0にしてください．インジケータ(pcirID)は，PCI拡張ROMに複数のイメージを格納するときに，このイメージを最後以外に配置する場合は，80h(ビット7を"1")とします．一つのイメージしか格納しない場合や複数のイメージの中の最後に配置する場合は，0とします．ユーザープログラム中で設定が必要なパラメータの一覧を表3に示します．

● アセンブル

作成したPCI拡張ROMプログラムを，MASMで通常のDOSプログラムのアセンブル同様に，アセンブルします．

```
>MASM SAMPLE[リターン]
```

この段階でインクルードしたOPTROM.INCにしたがい，ユーザー定義のパラメータなど各種ヘッダが正しく生成されます．アセンブルが終了すると，EXEファイルが出力されるはずです．

● ROM化ファイル生成ツールMAKEROM.COM

出力されたEXEファイルは，当然ながらそのままではROM化できません．EXEファイルからROM化可能なバイナリファイルを生成するのがこのMAKEROM.COMです．

```
>MAKEROM ?????.EXE ????.???[リターン]
```

最初のパラメータが変換元のEXEファイル名，次のパラメータがROM化のための出力ファイル名です．これにより，PCI拡張ROMに必要な初期化サイズ，イメージサイズ，そして正

Column 1

ROM化ファイル分割ツールROMDVI.COM

おまけとして，ROM化ファイルを偶数バイト/奇数バイトの2分割に，または1バイト目/2バイト目/3バイト目/4バイト目に4分割できるツール`ROMDVI.COM`を用意しました．

PCI拡張ROMは8ビットサイズのROMを使うのが一般的ですが，PCIバスは32ビットバスなので，PCI拡張ROMに対応したPCIデバイスにはバス幅変換の機構が必要です．このバス幅変換のための回路がPCIデバイス内に入りきらない場合は，PCI拡張ROMを16ビット幅や32ビット幅で使う場合もあるかもしれません．その場合，8ビット幅のROMを2個や4個ならべてPCI拡張ROMを実現する方法もあるので，そのときこのツールを使います．

8ビット幅のROMだけでPCI拡張ROMを構成する場合は，このツールを使う必要はありません．

しいチェックサムが書き込まれ，イメージサイズのバイト数のファイルサイズに調整されます．PCI拡張ROMの内容がどんな内容でも，全バイトを加算した結果を0にできるように，イメージの最後にチェックサムバイトを用意して，これでチェックサムを調整しています．よって実際に必要なイメージサイズがちょうど512バイトで割り切れるサイズだった場合は，チェックサムバイトをさらに追加するので，もう512バイト分増やして最終イメージサイズとしています．なお初期化サイズとイメージサイズは同じサイズにして処理しています．

〔表3〕ユーザープログラムで設定するパラメータ

パラメータ名	定義部分	
strROMData CATSTR	<'???????>	デバイス固有領域 最大18バイト）
strROMName CATSTR	<'?????',0>	PCIボード名/拡張ROMバージョンなど（任意サイズ）
pcirVendorID	= ????h	ベンダID
pcirDeviceID	= ????h	デバイスID
pcirVPD	= ????h	バイタル製品データへのポインタ
pcirClassBase	= ??h	基本クラスコード
pcirClassSub	= ??h	サブクラスコード
pcirClassProg	= ??h	プログラムインターフェース
pcirRevLvl	= ????h	拡張ROMリビジョン
pcirCode	= 00h	コードタイプ（0にする）
pcirID	= ??h	インジケータ

このツールで生成されるのは一つのイメージファイルだけです．複数のイメージを一つのROMに格納する場合は，このツールなどで作成したイメージファイルを連結して一つのファイルを作成し，それをROM化する必要があります．

こうして作成したROM化ファイルをROMライタで書き込み，PCI拡張ROMとします．

5 PCI拡張ROMサンプルプログラム

ここではサンプルとして，起動時にメッセージを表示したり，各種パラメータを設定するメニューを備えたPCI拡張ROMプログラムを作成してみました．

● PCI拡張ROMサンプルプログラムSAMPLE.ASM

リスト2に，PCI拡張ROMサンプルプログラム`SAMPLE.ASM`

〔リスト2〕PCI拡張ROMサンプルプログラムSAMPLE.ASM

```
INCLUDE OPTROM.INC
;************************************************
;
;x86 PC/AT互換機用 PCI拡張ROMサンプルプログラム
;
;************************************************
CODE1 SEGMENT PAGE COMMON 'CODE1'
.386p

    ASSUME  CS:CODE1,DS:NOTHING,ES:NOTHING

;****************************************
;PCIボード拡張ROM ヘッダ記述部
;****************************************
strROMData CATSTR <'PCIBoard OptionROM'>      ;デバイス固有領域(最大18バイト)
strROMName CATSTR <'Option ROM Sample Program Image ( for PC/AT ROM )',0> ;PCIボード名/拡張ROMバージョンなど
pcirVendorID = 6809h
pcirDeviceID = 8000h
pcirVPD      = 0000h
pcirClassBase = 05h
pcirClassSub  = 00h
pcirClassProg = 00h
pcirRevLvl    = 0000h
pcirCode      = 00h
pcirID        = 00h
initCodeLabel CATSTR <postCode> ←Ⓐ

        ROM_HEADER
```

〔リスト 2〕PCI 拡張 ROM サンプルプログラム SAMPLE.ASM（つづき）

```
MODE    DB      1
ADR     DW      1234h
STRS    DB      'ABCDEFGHIJKLMNOP'

;**************************************************
;
;x86 PC/AT 互換機用 PCI 拡張 ROMプログラム本体 開始
;
;**************************************************
;***************************
;
; 初期化コード
;
;***************************
postCode PROC FAR ◄──Ⓑ

        PUSH    CS
        POP     DS
    ASSUME  DS:CODE1

        MOV     SI,OFFSET TitleMsg
        CALL    dispTTY
;
; 30 秒間キー入力待ち
;
        MOV     DX,30
wait1s:
        MOV     CX,33333    ;33333*(15*2)us = 1s
wait15us1:
        IN      AL,61h
        TEST    AL,10h
        JZ      wait15us1
wait15us2:
        IN      AL,61h
        TEST    AL,10h
        JNZ     wait15us2

        MOV     AH,1
        INT     16h
        JNZ     getKey

        LOOP    wait15us1
        DEC     DX
        JNZ     wait1s
        JMP     exit
getKey:
        MOV     AH,0
        INT     16H
        CMP     AX,2C00H
        JNE     exit

        CALL    menu

exit:

;                                              ┐
; 初期化コード部を ROM SIZE から削る            │
;                                              │
        MOV     CX, OFFSET postCode+1          │
        ADD     CX, 511 ;1023                  │
        SHR     CX, 9                          │
        MOV     [ROM_SIZE], CL                 │
        MOV     BYTE PTR [postCode], 0CBH ;RETF│
                                               │
        SHL     CX, 9                          │
        DEC     CX                             │
        XOR     AL, AL                         │
        XOR     SI, SI                         ├Ⓒ
calcChkSumLoop:                                │
        SUB     AL, [SI]                       │
        INC     SI                             │
        LOOP    calcChkSumLoop                 │
                                               │
        MOV     [SI], AL                       │
                                               │
                                               │
;        MOV     AX,4C00H ┐                    │
;        INT     21H      ┘Ⓓ                   │
        RET                                    │
                                               │
postCode ENDP                                  ┘
```

```
;***************************
;
; Menu 表示
;
;***************************
menu PROC NEAR

        MOV     AH,03H
        XOR     BH,BH
        INT     10H
        MOV     [cursorShape],CX
;
; Copy initial item value
;
        MOV     AL, [MODE]
        MOV     [curMode], AL
             ∫
           途中略
             ∫
        MOV     CX, 2000H
        INT     10H
        POP     CX
        POP     AX
        RET

clearCursor ENDP

TitleMsg DB 'Option ROM Sample Program',13,10
         DB 'Press any key to continue.',13,10
         DB '(ALT-Z is start up the menu)',13,10,0

MenuTitleMsg DB 'PCI Option BIOS ROM Menu Sample Program',0
MenuKeyMsg   DB 18h,19h,':Select Items  Enter:Change Item',0

QuitMsg    DB 'Save Parameters ?(Y/N)',0
QuitKeyMsg DB 'Y:Exit and save changes N:Exit and discard changes
                                                              Esc:Cancel',0

ItemInfo STRUC
 ItemMsg  DW ?
 ItemDisp DW ?
 ItemEdit DW ?
 ItemUpdate DW ?
ItemInfo ENDS

ItemTable LABEL ItemInfo
          ItemInfo<ModeMsg,ModeDisp,ModeEdit,ModeUpdate>
          ItemInfo<AdrsMsg,AdrsDisp,AdrsEdit,AdrsUpdate>
          ItemInfo<StrMsg,StrDisp,StrEdit,StrUpdate>
          ItemInfo<ExitMsg,ExitDisp,ExitEdit,ExitUpdate>
MAX_ITEM_NUM EQU ($-ItemTable) / (SIZE ItemInfo)
          DW 0

ModeMsg   DB 'Mode',0,'                     [0..9  Default:1]',0
AdrsMsg   DB 'Address',0,'      h           [0000h..FFFFh  Default:1234h]',0
StrMsg    DB "Strings",0,"  '               '
                            [Max 16 Characters Default:'ABCDEFGHIJKLMNOP']",0
ExitMsg   DB 'Exit',0,0

cursorShape DW ?
enterMode  DB 0
enterLen   DW 0
@ENTER_STR   EQU 0
@ENTER_HEX   EQU 1
@ENTER_NUM   EQU 2
curItemNum DB 0
curMode    DB ?
curAdrs    DW ?
curStr     DB 16 DUP(?)
strBuf     DB 16 DUP(?)
           DB 0
;***************************************************************************
;
;x86 PC/AT 用拡張 ROMプログラム本体終了
;
;***************************************************************************
CODE1 ENDS
      END ROM_JMP
```

を示します．

ソースそのものは通常のアセンブラでアセンブルするので，ソースの記述方法は通常のアセンブラプログラムと同じです．まずOPTROM.INCをインクルードします．そして各種ヘッダの内容を定義します．PCI拡張ROMヘッダの後には，メニューで設定する各種パラメータを格納しています．

● プログラムの流れ

まず最初にFAR CALLされるのは，PCI拡張ROMヘッダのオフセット3のアドレスですが，リスト1でわかるようにここは，

```
JMP  NEAR PTR &initCodeLabel
```

と記述されています．さらにリスト2のSAMPLE.ASM上ではⒶで，

```
initCodeLabel CATSTR <postCode>
```

としているので，ラベルpostCodeが定義されているⒷにジャンプしてくることになります．ここから実際のプログラムを記述するわけです．

サンプルプログラムでは，まずPCI拡張ROMサンプルプログラムのタイトルを表示し，約30秒間のキー入力待ちループに入ります．キー入力があれば拡張ROMメニューを表示して，拡張ROMメニューの制御に移ります．約30秒間キー入力がない，もしくはキー入力があってもALT+Zキー（キーコードAX=2C00h）でない場合は，PCI拡張ROMプログラムの終了処理に入ります．

リスト2のⒸの部分が終了処理です．ここでは，初期化サイズからランタイムサイズに変更するための処理を行い，削減したサイズをPCI拡張ROMヘッダの初期化サイズと，PCIデータ構造中のイメージサイズに書き込んでいます．また，チェックサムを再計算して，イメージサイズの最後のバイトをチェックサムバイトとして，再計算した結果を書き込みます．あとはRET命令を実行してBIOSに戻るだけです．

なお，拡張ROMメニューそのものは，PC/ATのBIOSコールによりキー入力を判定したり画面を書き換えているだけなので，ここでは説明を省略します．アセンブラの入門書とPC/ATのBIOSの使い方の解説書を参考にしてください．

● メニュープログラム開発方法の一例

PCI拡張ROMプログラムのデバッグには手間がかかります．システム起動時にしか実行されないので，何度もリセットを繰り返して動作を試すことになるからです．PCI拡張ROM内の処理の一部でも，通常動作時にアルゴリズムの確認レベル程度のデバッグができると，開発効率があがります．

今回開発したサンプルプログラムでは，とくにメニュー部分の動作のデバッグが問題となるでしょう．このような，基本的には初期化処理とは関係ない部分は，その部分だけを通常のDOSプログラムとして動かすことでデバッグを行うことができます．

SAMPLE.ASM中のⒹの2行はコメントアウトされています．じつはこのコメントを削除すると，生成したEXEファイルをなんとそのままDOSプログラムとして実行できてしまいます．このコメントアウトされている2行は，DOSのファンクションコールで，プログラムの終了時の処理に相当します．

これによりメニューが正しく動作するかなどの確認がDOSコマンドとして行えるようになります．最終的に拡張ROMプログラムとするときは，これをコメントしてRET命令によりシステムにリターンするようにするわけです．

もちろん，EXEプログラムとしてDOS環境で動かす場合は，AXレジスタに格納されているはずのPCIバス番号やデバイス番号などが格納されていない状態でコールされるので，これらのパラメータを利用するPCI拡張ROMプログラムの場合は，ダミーの値をプログラム中に埋めておくなどの対処が必要です．

● サンプルのROM化

アセンブル後のファイルSAMPLE.EXEから，MAKEROM.COMによりイメージファイルSAMPLE.ROMを生成させます．

```
>MAKEROM SAMPLE.EXE SAMPLE.ROM［リターン］
```

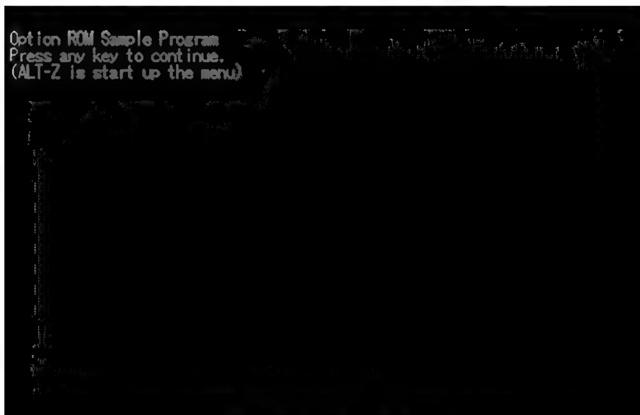
ここではPCI拡張ROMに一つのイメージのみ格納するので，これで作成したSAMPLE.ROMをそのままROM化してください．

図3に作成したSAMPLE.ROMのダンプイメージを示します．ヘッダの並び，オフセット値などを表1および表2と照らし合

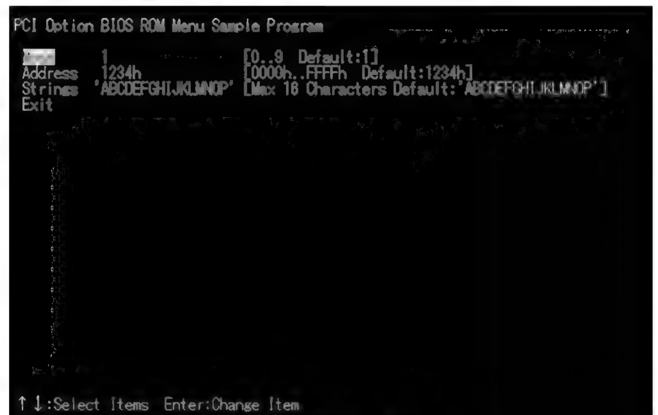
〔図3〕SAMPLE.ROMのダンプ

```
      +0 +1 +2 +3 +4 +5 +6 +7   +8 +9 +A +B +C +D +E +F | ---- ASCII -----
000h  55 AA 04 EB 72 90 50 43 - 49 42 6F 61 72 64 20 4F | U...r.PCIBoard O
010h  70 74 69 6F 6E 52 4F 4D - 4C 00 4F 70 74 69 6F 6E | ptionROML.Option
020h  20 52 4F 4D 20 53 61 6D - 70 6C 65 20 50 72 6F 67 |  ROM Sample Prog
030h  72 61 6D 20 49 6D 61 67 - 65 20 28 20 66 6F 72 20 | ram Image ( for
040h  50 43 2F 41 54 20 52 4F - 4D 20 29 00 50 43 49 52 | PC/AT ROM ).PCIR
050h  09 68 00 80 00 00 18 00 - 00 05 00 00 04 00 00 00 | .h..............
060h  00 00 00 00 01 34 12 41 - 42 43 44 45 46 47 48 49 | .....4.ABCDEFGHI
070h  4A 4B 4C 4D 4E 4F 50 0E - 1F BE 0B 05 E8 54 04 BA | JKLMNOP......T..
080h  1E 00 B9 35 82 E4 61 A8 - 10 74 FA E4 61 A8 10 75 | ...5..a..t..a..u
090h  FA B4 01 CD 16 0F 85 08 - 00 E2 EA 4A 75 E4 EB 0F | ..........Ju...
0A0h  90 B4 00 CD 16 3D 00 2C - 0F 85 03 00 E8 23 00 B9 | .....=.,.....#..
0B0h  78 00 81 C1 FF 01 C1 E9 - 09 88 0E 02 00 C6 06 77 | x..............w
0C0h  00 CB C1 E1 09 49 32 C0 - 33 F6 2A 04 46 E2 FB 88 | .....I2.3.*.F...
0D0h  04 CB B4 03 32 FF CD 10 - 89 0E E0 06 A0 64 00 A2 | ....2........d..
0E0h  E6 06 A1 65 00 A3 E7 06 - B9 10 00 BE 67 00 BF E9 | ...e........g...
0F0h  06 8A 04 88 05 46 47 E2 - F8 E8 03 04 E8 7E 03 B3 | .....FG......~..
                          ～以下略～
```

〔図4〕PCI 拡張 ROM サンプルプログラムの動作例



（a）システム起動時



（b）メニュー起動時

わせながら見てください.

● PCI 拡張 ROM サンプルプログラムの動作

図4に，ROM 化した PCI 拡張 ROM サンプルプログラムを，PCI 拡張 ROM 対応ボードに実装してシステムを起動したときの初期化画面の例を示します〔図4（a）〕. メッセージが表示され約30秒間のキー入力待ち状態に入ります. このとき ALT+Z キーを押すと PCI 拡張 ROM のメニューが起動します. ALT+Z キー以外のキーが入力された場合や，約30秒キー入力がなければ，キー入力待ちを抜けてシステムの起動処理に戻ります.

PCI 拡張 ROM のメニューでは，モード，アドレス，パラメータの3種類の設定が可能です〔図4（b）〕. カーソルキーで項目を移動して任意の値に設定することが可能です. Exit を選択すると，破棄して終了するか，保存して終了するかを選択します. どちらかを選択すると，メニューをクリアしてシステムの起動処理に戻ります. 破棄して終了の場合は，デフォルトの値がそのまま拡張 BIOS 領域に保存されます. 保存して終了の場合は，メニューで設定した値が拡張 BIOS 領域に保存されます.

6 PCI 拡張 ROM の活用例

● 本来の PCI 拡張 ROM の使い方

たとえば PCI 拡張 ROM 搭載の SCSI ボードでは，内蔵の IDE HDD と同じように INT 13h の BIOS コールを使って，ブロック単位で SCSI HDD の読み書きができます（もっとも，ROM がなくてもドライバを組み込めば同様のことが可能だが）. これは SCSI ボードに搭載されている PCI 拡張 ROM がシステム標準の BIOS の機能を拡張し，内蔵の IDE HDD と同様のプログラミングインターフェースで SCSI HDD も読み書きできるようにしているためです.

ここまで本格的な応用例は準備できませんが，PCI 拡張 ROM メニューで設定したパラメータを，DOS 上で取得するサンプルプログラムを作成してみました. PCI 拡張 ROM の機能を DOS 上から使う一例として参考にしてください.

● PCI デバイス/ボードの存在確認方法

まず，制御したい PCI デバイスやボードがそのシステムに実装されているかを確認する必要があります. これまでの PCI デバイスのデバッグプログラムのように，ベンダ ID やデバイス ID によって PCI デバイスを検索するのも一つの方法ですが，それだけでは問題があります.

すでに説明したように，PCI デバイスに PCI 拡張 ROM の実装機能があっても，本当に ROM が実装されていなければ，PCI

Column 2

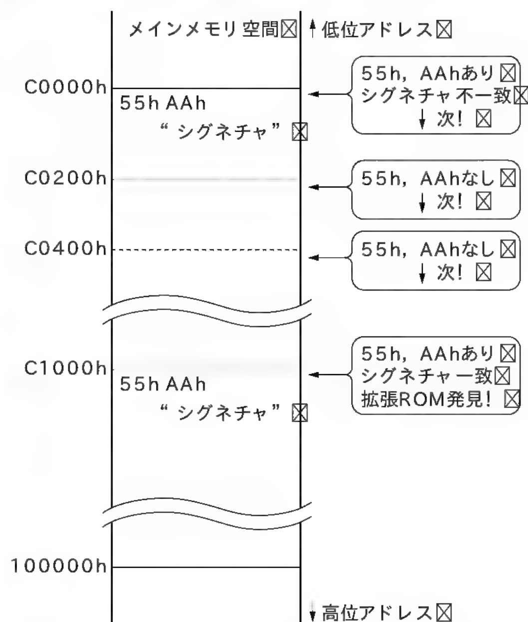
初期化時だけ必要な場合/ランタイムだけ必要な場合

PCI デバイスによっては，初期化処理だけを行った後はいっさい PCI 拡張 ROM が必要ない場合や，逆に初期化処理は必要ないが，ランタイムとして拡張プログラムだけを実装したい場合もあるでしょう.

初期化時だけ必要な場合は，POST 時に BIOS からコールされて初期化プログラムを実行したあと，BIOS に返す前に PCI 拡張 ROM ヘッダや PCI データ構造の初期化サイズやイメージサイズを 0 にクリアします. すると BIOS はランタイムが必要ないものと理解して，そのイメージを拡張 BIOS 領域に残しません. 1 バイトも漏らさず，完全にメモリ上から消えることになります.

ランタイムだけ必要な場合は，POST 時に BIOS からコールされた場合，何もせずに RET 命令で戻ります. PCI 拡張 ROM ヘッダや PCI データ構造の初期化サイズやイメージサイズを書き換えていないので，PCI 拡張 ROM 内に実装しているときのまま，ランタイム時も拡張 BIOS 領域にイメージが確保され，残ります.

〔図5〕拡張BIOS領域の検索方法



拡張ROMは存在しないも同じです．つまり，PCI拡張ROMの内容がシャドウRAM上に転送されているかどうかを確認することで，制御したいPCIデバイスやボードが正しくシステムに実装されているかを判断します．

● 拡張BIOS領域のPCI拡張ROMの検索

PCI拡張ROMの内容は，C0000h以降の拡張BIOS領域のどこかに転送されます．ここからFFFFFhまでのアドレスを512バイト単位でサーチします．イメージサイズが512バイト単位なので，まずはC0000hを，次にC0200hを，次にC0400hを，…というように調べていくわけです（図5）．

該当するPCI拡張ROMかどうかの判定方法としては，PCI拡張ROMプログラムの起動シーケンスで説明したように，PCI拡張ROMヘッダの先頭が55h AAhかどうか，PCIデータ構造シグネチャがあるかなどを一つずつ確認する方法があります．この場合はベンダIDやデバイスID，さらにクラスコードまで判定すれば十分でしょう．

もう一つ，いちばん簡単で確実な方法は，PCI拡張ROMヘッダの先頭が55h AAhであることを見つけたら，その後ろに，PCI拡張ROM内のデバイス固有の情報が格納されているかをサーチすることです．ここで用意したPCI拡張ROMサンプルプログラムでは，PCI拡張ROMヘッダとPCIデータ構造の間に“Option ROM Sample Program Image (for PC/AT ROM)”といった文字列を格納しています．ここにメーカ名や製品名を入れておけば，ほぼ間違いなく，該当のPCIデバイス/ボードであるかを判定できます（もちろんそのためには，ありがちな名前や名称ではなく，ユニークな文字列であることが望ましい）．つまり，ほかでは絶対使っていないはずだという文字列が拡張BIOS領域に存在しているかどうかを確認する，データ一致検索プログラムを組めば良いわけです．

なお，PCI拡張ROMサンプルプログラムでは，固有文字列の最後にNULLを格納しているので，C言語標準の文字列一致検査関数などを使って調べることも可能です．

● サーチプログラム2種

ここではアセンブラ（リスト3）とC言語（リスト4）によるサーチプログラムの2本を作成しました．なおアセンブラ版では，ヘッダ定義ファイルOPTROM.INCをインクルードしてここでも活用しています．

まずどちらも，C0000hから1Mバイト未満の領域内を，512バイトごとにアクセスして2バイト読み出し，55h AAhであるかどうかを調べます．

その後，アセンブラ版では，先頭からオフセット1Ahのところから固有文字列「“Option ROM Sample Program Image (for PC/AT ROM)”」が存在しているかをチェックします．C言語版では，PCIデータ構造シグネチャを調べ，ベンダID，デバイスIDを確認したあとで固有文字列が一致するかを調べています．

PCI拡張ROMの転送先アドレスを発見したら，先頭からオフセット64hのアドレスをバイトサイズで読み出しモード値として，オフセット65hをワードサイズで読み出してアドレス値として，オフセット67hから16バイトをパラメータ文字列として読み出し，表示します．

マシンをリセット起動して，PCI拡張ROMのメニューを起

〔リスト3〕アセンブラ版拡張ROMサーチプログラム

```
INCLUDE  OPTROM.INC
.386
CODE SEGMENT BYTE PUBLIC 'CODE' USE16

    ASSUME  CS:CODE,DS:CODE,ES:CODE


    ORG  81h
CMDLIN   LABEL     BYTE

    ORG  100h

START    PROC NEAR
```

```
;
; Display Title Message
;
    MOV  DX, OFFSET titleMsg
    MOV  AH, 9
    INT  21H
;
; Search Option ROM
;
    MOV  DX, 0C000h
tryROMSearch:
    MOV  ES, DX
```

〔リスト 3〕アセンブラ版拡張 ROM サーチプログラム（つづき）

```
    ASSUME  ES:NOTHING
    ;
    ; Check ROM ID
    ;
    MOV  CX, 512/16
    CMP  WORD PTR ES:[0],0AA55h
    JNE  nextUMB        ; Jump if not Option ROM ID

    ;
    ; Compare idntify string
    ;
    MOV  CL, ES:[2]
    XOR  CH, CH
    SHL  CX, 9-4
    PUSH CX
    MOV  DI, 18h+2
    MOV  SI, OFFSET searchMsg
    MOV  CX, sizeSearchMsg
    CLD
    REP  CMPS BYTE PTR DS:[SI],BYTE PTR ES:[DI]
    POP  CX
    JNE  nextUMB        ; Jump if unmatch identify string

    MOV  [flag], 1
;
; Display Parameters
;
    PUSH CX
    PUSH DX
    ;
    ; Make ROM address string
    ;
    MOV  DI, OFFSET romAdrsStr
    CALL binToHex4

    ;
    ; Make Mode String
    ;
    MOV  SI, ES:[18h]
    ADD  SI, ES:[SI].PCIR_LEN
    MOV  AL, '0'
    ADD  AL, ES:[SI]
    MOV  [modeStr], AL

    ;
    ; Make Address String
    ;
    MOV  DX, ES:[SI+1]
    MOV  DI, OFFSET adrStr
    CALL binToHex4

    ;
    ; Copy String
    ;
    MOV  DI, OFFSET strStr
    ADD  SI, 3
    MOV  CX, 16
copyString:
    MOV  AL, ES:[SI]
    TEST AL, AL
    JZ   endString
    MOV  [DI], AL
    INC  DI
    INC  SI
    LOOP copyString
endString:
    MOV  BYTE PTR [DI], "'"
    MOV  WORD PTR [DI+1], 0A0Dh
    MOV  WORD PTR [DI+3], '$'

    MOV  DX, OFFSET paramMsg
    MOV  AH, 9
    INT  21h
    POP  DX
    POP  CX
nextUMB:
    ADD  DX, CX
    JC   exit       ; Jump if over option ROM area
    CMP  DX, 0F000h
    JB   tryROMSearch   ; Try next ROM area
exit:
    CMP  [flag], 0
    JNE  foundROM
    MOV  DX, OFFSET notFoundMsg
    MOV  AH, 9
    INT  21h
foundROM:
    RET

START   ENDP
;*********************************
;
; Convert to Hex string
;
;  Entry:
;   DX   Convert value
;   DS:DI     Convert string store address
;  Exit:
;   none
;  Modify:
;   AL, CX
;*********************************
binToHex4     PROC NEAR

    MOV  CX, 4
nextAdrs:
    ROL  DX, 4
    MOV  AL, DL
    AND  AL, 0Fh
    ADD  AL, '0'
    CMP  AL, 10+'0'
    JB   storeAdrs
    ADD  AL, 'A'-'0'-10
storeAdrs:
    MOV  [DI], AL
    INC  DI
    LOOP nextAdrs
    RET

binToHex4     ENDP

flag          DB   0

titleMsg DB   'PCI Option BIOS ROM Search ASM Program Ver1.0 by SUGAWARA',13,10,'$'

paramMsg DB   'Option ROM Found',13,10
         DB   '  ROM Address = '
romAdrsStr    DB   '00000h',13,10
         DB   '  Parameters',13,10
         DB   '    Mode    = '
modeStr       DB   '0',13,10
         DB   '    Address = '
adrStr        DB   '0000h',13,10
         DB   "    Strings = '"
strStr        DB   "0123456789ABCDEF'",13,10,'$'

notFoundMsg   DB   'Not found option ROM',13,10,'$'

searchMsg     DB   'Option ROM Sample Program Image ( for PC/AT ROM )',0
sizeSearchMsg = $ - searchMsg

CODE     ENDS
    END  START
```

〔リスト4〕C言語版 拡張ROMサーチプログラム

```
/*

 PCI拡張ROM設定パラメータ　取得サンプルプログラム for DOS
     search_c.c

*/

#include <stdio.h>
#include <string.h>
#include <dos.h>

main()
{
    int i,l;
    unsigned int SegAdr,VndID,DevID;
    short int far *ExpRomWord;
    char far *ExpRomByte;
    char PCIData[]="PCIR";
    int PCIDataOff,BoardNameOff,BoardParaOff;
    char mode;
    unsigned int address;
    char PCIExpRomString[]=
             "Option ROM Sample Program Image ( for PC/AT ROM )";


    printf("PCI Option BIOS ROM Search C Program Ver1.0
                                                   by SUGAWARA¥n");

    /* 拡張ROM格納領域　先頭アドレス */
    SegAdr=0xc000; /* セグメント */

    /* PCIボード名/拡張ROMバージョン文字列格納オフセット */
    BoardNameOff=0x1a;
    /* 設定パラメータ格納オフセット */
    BoardParaOff=0x64;

    l=strlen(&PCIExpRomString[0]);

    while(1){

         /* 拡張ROMヘッダ検索 */
         ExpRomWord=MK_FP(SegAdr,0);  /* オフセット 0 */
         if (*ExpRomWord == 0xaa55) { /* 55h AAh 拡張ROM確認 */

              /* PCIデータ構造シグネチャチェック      */
              /* PCIデータ構造へのポインタ格納アドレス */
              ExpRomWord=MK_FP(SegAdr,0x18);
              PCIDataOff=*ExpRomWord;
              /* PCIデータ構造先頭アドレス */
              ExpRomByte=MK_FP(SegAdr,PCIDataOff);
              for(i=0;i<4;i++){   /* "PCIR"文字列チェック */
                   if (*ExpRomByte != PCIData[i]) break;
                   ExpRomByte++;
              }
              if (i==4) {    /* PCI拡張ROM確認 */

                   /* ベンダID/デバイスID チェック */
                   /* ベンダID格納アドレス         */
                   ExpRomWord=MK_FP(SegAdr,PCIDataOff+4);
                   VndID=*ExpRomWord;
                   ExpRomWord++;  /* デバイスID格納アドレス */
                   DevID=*ExpRomWord;
                   /* ベンダID&デバイスID一致 */
                   if ((VndID==0x6809) & (DevID==0x8000)) {

                        /* PCIボード名/拡張ROMバージョンなど */
                        /* 文字列チェック                    */
                        /* シグネチャ文字列オフセット        */
                        ExpRomByte=MK_FP(SegAdr,BoardNameOff);
                        /* 拡張ROMに埋め込んでいる文字列チェック */
                        for(i=0;i<l;i++){
                             if (*ExpRomByte !=
                                        PCIExpRomString[i]) break;
                             ExpRomByte++;
                        }
                        if (i==l) break;

                   }

              }

         }

         /* 512バイト単位で検索 次のアドレス */
         SegAdr=SegAdr+512/16;    /* セグメントは/16 */
         if (SegAdr < 0xc000) {  /* アドレス1Mバイトを超えた */
              printf("Not found option ROM¥n");
              return(-1);
         }
    }

    /* PCI拡張ROM発見 */
    printf("Option ROM Found¥n  ROM Address = %04X0h¥n",SegAdr);
    printf("  Parameters¥n");

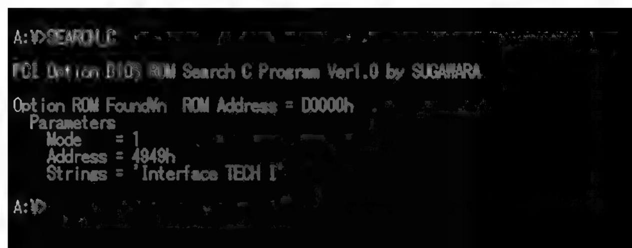
    /* モード */
    /* 設定パラメータ格納オフセット */
    ExpRomByte=MK_FP(SegAdr,BoardParaOff);
    mode=*ExpRomByte;
    printf("    Mode     = %d¥n",mode);
    ExpRomByte++;

    /* アドレス */
    address=*ExpRomByte;                      /* 下位アドレス */
    ExpRomByte++;
    address=address+((*ExpRomByte)<<8);       /* 上位アドレス */
    printf("    Address = %04Xh¥n",address);
    ExpRomByte++;

    /* 文字列 */
    printf("    Strings = '");
    for(i=0;i<16;i++){
         printf("%c",*ExpRomByte);
         ExpRomByte++;
    }
    printf("'¥n");
}
```

動して各パラメータを書き換えて保存&終了し，DOSを起動してこれらのサーチプログラムを実行してみてください．メニューで設定した値が，DOS環境で正しく取得できることがわかると思います（図6）．

〔図6〕サーチ結果の表示画面



参考文献
1)『PCI Local Bus Specification Rev2.1 and Rev2.3』，PCI-SIG
2)「PCIバスの詳細と応用へのステップ」，『OPENDESIGN』，No.7，CQ出版（株）

すがわら・なおのぶ　BIOS/ファームウェアプログラマ

Column 3

PCI拡張ROM対応デバイス

PCI拡張ROMは，特殊なバスコマンドでアクセスする空間ではありません．ハードウェア的には通常のメモリ空間とまったく同じです．若干異なるのは，通常のメモリ空間はベースアドレスレジスタ0～5を使うものの，PCI拡張ROMの場合は，コンフィグレーションレジスタ空間の30hに専用のベースアドレスレジスタをもっている点です．

PCIデバイスにPCI拡張ROMを実装するには，まずコンフィグレーションレジスタにPCI拡張ROMベースアドレスレジスタを実装する必要があります．実装するフリップフロップはビット11までなので，4Kバイト単位のアドレスを先頭アドレスとして配置できることになります．またビット0がPCI拡張ROMイネーブルビットで，PCI拡張ROMにアクセスするにはこのビットもセットされていないとアクセスできません．

リストAとリストBに，PCI拡張ROM対応デバイスのVHDLソースの一部を示します．コンフィグレーションレジスタの30hのところに，レジスタを読み書きできるように追加します．またアドレスデコード部では，バスコマンドがメモリアクセスコマンドかどうか，アドレスが一致しているか，さらにPCI拡張ROMイネーブルビットがセットされているか，そしてコマンドレジスタのメモリ空間イネーブルビットもセットされているかどうかもデコード条件に含めます．以上の条件が成立してはじめてPCI拡張ROMへのアクセスが可能になります．あとは通常のメモリをPCIメモリ空間に実装するのと同様のハードウェア構成です．

この空間は拡張ROM空間ですが，この空間に対してCPUが書き込み動作を行えば，PCIバス上ではメモリライトコマンドが発行されます．メモリライトコマンドが発行された場合は，正しくDEVSEL応答をしてバスアクセスを正常に終了させるようにし，内部的には書き込みデータを無視するようにすればよいでしょう．

ちなみに筆者は，拡張ROMをデバッグする際は，ハードウェア的にはこの空間をRAMとして読み書きできるようにし，プログラムを修正したROMイメージをPCI側から書き込み，リセットしてシステムを再起動させて動作確認を行うなどしています．またプログラム中にハングアップするようなバグがあった場合でも，拡張ROMを無視してシステムを起動できるように，PCIデバイスの外から拡張ROMベースアドレスレジスタを無効化できるようなスイッチを実装して，いちいち開発中のPCIボードを外さなくてもデバッグを続行できるような工夫もしています．

なお，拡張ROMをPCIデバイスの外に実装する場合，実際にはデータバス幅8ビットや16ビットのROMを実装することが多いでしょう．その場合，PCIバスのデータバスは32ビットなので，16ビット幅のROMなら2回，8ビット幅のROMなら4回に分けてアクセスして，32ビット分のデータを用意してPCI側に返さなければなりません．

山武 一朗　来栖川電工有限会社

〔リストB〕PCI拡張ROM対応デバイスのアドレスデコーダ

```
-- ********** アドレスデコーダ ***** --
Address_Decoder : process (
        ～中略～
    -- 拡張ROM空間へのアクセスかどうかを認識
    if (
            -- コンフィグレーションサイクル
            PCI_BusCommand(3 downto 1) = PCI_MemCycle
        ) and (
            CFG_Cmd_Mem = '1'   -- メモリ空間イネーブルビット
        ) and (
            CFG_ExpROM_En = '1'-- 拡張ROMイネーブルビット
        ) and (
            PCI_Address(31 downto 20) =
                            CFG_ExpROM_Addr(31 downto 20)
        )
    then
        Hit_ExpROM <= '1'; -- 拡張ROMサイクルヒット
    else
        Hit_ExpROM <= '0';
    end if;

end process Address_Decoder;
```

〔リストA〕PCI拡張ROM対応デバイスのローカルバスシーケンサ

```
-- ********** ローカルバスシーケンサ ステートマシン ********** --
        ～中略～
-- ********** LOCAL_CFG_ACCESS時の動作 ********** --
when LOCAL_CFG_ACCESS =>    -- コンフィグレーションサイクル

-- コンフィグレーションライトサイクル
if (PCI_BusCommand(0) = '1' ) then

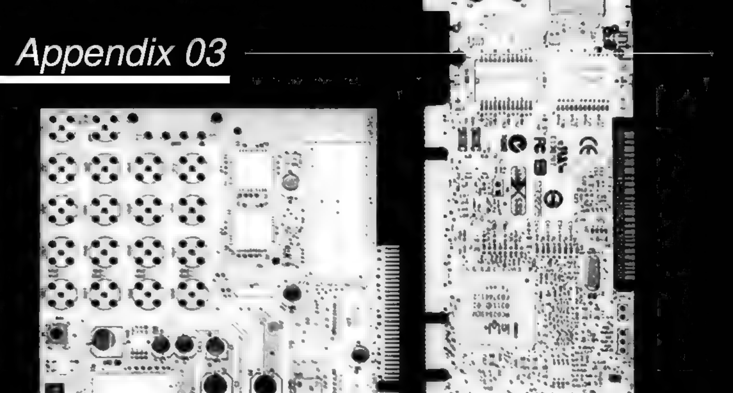
    case PCI_Address(7 downto 2) is
        ～中略～
        when "001100" =>    -- 拡張ROMベースアドレスレジスタ(+30h)
            if (C_nBE(3) = '0') then
                CFG_ExpROM_Addr(31 downto 24)
                                <= PCIAD(31 downto 24);
            end if;
            if (C_nBE(2) = '0') then
                CFG_ExpROM_Addr(23 downto 20)
                                <= PCIAD(23 downto 20);
            end if;
            if (C_nBE(0) = '0') then
                CFG_ExpROM_En <= PCIAD(0);
            end if;

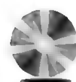
        when others => null;
    end case;

else    -- コンフィグレーションリードサイクル

    case PCI_Address(7 downto 2) is
        ～中略～
        when "001100" =>    -- 拡張ROMベースアドレスレジスタ
            PCIAD_Port(31 downto 20) <= CFG_ExpROM_Addr;
            PCIAD_Port(19 downto  1) <= (others => '0');
            PCIAD_Port(0) <= CFG_ExpROM_En;
        ～中略～
    end case;

end if;
```





PCIデバッグライブラリ for DOS 新バージョン登場！

菅原 尚伸

● PCIデバッグライブラリ for DOSとは？

PCIデバイスを設計/開発中に，そのデバイスのメモリやレジスタが正しく読み書きできるかどうかを確認したいという場面は多々あります．このとき，いきなりWindows環境で動作確認を行うのはたいへん危険です．またWindows環境でハードウェアに対してアクセスするには，デバイスドライバも必要になります．

そこでもっと手軽に，MS-DOS環境でPCIデバイスが正しく動作するかどうかを確認できると便利です．しかしMS-DOS環境は16ビットモードなので，4Gバイトのアドレス空間をもつPCIを簡単には扱えません．そこで，16ビットモードと32ビットモードの壁を越えて，4Gバイトのメモリ空間にアクセスしたり，32ビットサイズでのI/Oアクセスを可能にするライブラリを用意しました．これが"PCIデバッグライブラリ for DOS"です．詳細については参考文献1)を参照してください．

今回説明する新バージョンのデバッグライブラリは，本誌の付属CD-ROMに収録しています．

● バーストリードサイクルを発生させたい！

従来のデバッグライブラリでは，PCIデバイスの動作確認に必要な各種バスアクセスを発生させる機能を実装していますが，このソフトウェアだけでは実現できなかったアクセスに，バースト転送発生機能があります．

とはいえ，メモリ→PCI方向のブロック転送関数などを使えばバーストライトサイクルが発生しました．しかしPCI→メモリ方向でブロック転送関数を使っても，バーストリードサイクルは発生しません．設計したPCIデバイスのメモリ空間に対して，バーストリードサイクルを発生させた場合の動作確認ができなかったのです．

なお，CPU転送でバースト転送が発生する理由については，参考文献1)の付属CD-ROMに収録されている記事を参照してください．

● 32ビットを超えるサイズのリード命令を実行すれば

一般的なPCIバスはデータバス幅が32ビットです．もしCPUがPCIメモリ空間で一度に32ビットを超える幅のサイズでリード命令を実行すれば，ホスト-PCIブリッジはそれをバーストリードサイクルに変換してくれるのではないかと考えました．

現在ではAMDから64ビット対応のx86系CPUが登場していますが，今なお使われているx86のほとんどはIA-32，つまり32ビットのx86です．よって，通常のメモリアクセスは32ビットまでしかありません．CPUの中で32ビットを越えるビット幅をもつレジスタといえば……そうです，FPUのレジスタとSIMD系命令があります．

そこで，ハイメモリアクセスライブラリに，FPU命令による64ビットアクセス，そしてSSE命令による128ビットアクセスのメモリアクセス関数を用意しました．リストAに関数定義のヘッダファイルを示します(MMXによる64ビットアクセス関数も実装している)．64ビットや128ビットでは変数の型がないので，リード/ライトのデータはバイト型の配列を8もしくは16バイト用意して，そこをバッファとしてデータをやり取りします．これにより，64ビットアクセス関数では2ワードバーストが，128ビットアクセス関数では4ワードバーストサイクルが発生します．

またFPUやSIMD系の命令は，すべてのCPUに実装されているわけではありません．現在動作しているCPUが各機能に対応しているかどうかを調べるために，`_getCPUType()`と`_getCPUID()`関数も用意しています．これらの関数の戻り値などについてはサンプルプログラムを参照してください．

リストBにハイメモリアクセスライブラリを使ったCプログラムのサンプルを，図Aに32ビットPCIに対して64ビットアクセスをした場合のバスの動きを示します．

● P6ファミリ以降のキャッシュ制御機能を使う

FPUはPentium以降で，MMXはMMX Pentium以降で使えますが，その次のPentiumPROの世代から，CPUのFSBの仕様が大きく変わり，それに合わせてキャッシュ制御機能が内蔵されました．これを使い，PCIメモリ空間をキャッシュ空間に指定することで，より多ワード数のバーストアクセスを発生させることができるはずです．

P6ファミリ以降のキャッシュ制御機能は，ベースアドレスとサイズ，その範囲のキャッシュモードを指定します．キャッシュモードには，アンキャッシュ，ライトスルー，ライトコンバイン(まとめて書き込む)，ライトプロテクト，ライトバックがあります．

また，複数の範囲を指定できるように，これらのレジスタセットが複数実装されています．CPUの種類によって，キャッシュ制御レジスタの本数が異なる場合があるかもしれないので，あらかじめインデックスが何番まであるかを確認しておきます．

リストCにキャッシュ制御関数のヘッダファイルを示します．`_getNumberOfCacheRangeIndex()`関数で，キャッシュ制御レジスタのインデックス本数を取得します．`_getCacheMode()`はキャッシュ制御レジスタの状態を取得，`_setCacheMode()`は設定する関数です．引き数としては，設定しようとするレジスタのインデックス番号，キャッシュ領域のベースアドレス，その空間サイズ，そしてキャッシュモードと，その設定を有効にするか禁止にするかの指定です．

ベースアドレスや空間サイズの指定はレジスタ幅が64ビットありますが，実際に値を設定できるのはビット36～12までです．なお，空間サイズの指定は，2の補数に変換した値を書き込みます．たとえば空間サイズを1Mバイト(0010_0000h)に

〔リストA〕64ビット/128ビット幅メモリアクセス関数

```
void _readHimem64(unsigned long Address, unsigned char *readBuffer);
void _writeHimem64(unsigned long Address,unsigned char *writeBuffer);
void _readHimem64MMX(unsigned long Address, unsigned char *readBuffer);
void _writeHimem64MMX(unsigned long Address,unsigned char *writeBuffer);
void _readHimem128(unsigned long Address, unsigned char *readBuffer);
void _writeHimem128(unsigned long Address,unsigned char *writeBuffer);

void _getCPUID(unsigned int, unsigned char *cpuidBuf);
unsigned int _getCPUType();
```

〔リスト B〕ハイメモリアクセスライブラリを使ったサンプルプログラム

```
/*
    MEMFUNC sample
*/

#include <stdio.h>
#include "memfunc.h"

main()
{
    int j,cpu;
    unsigned char c;
    unsigned int  i;
    unsigned long l;
    unsigned char data[1024];

    /* 動作CPU環境確認 */
    i=_getCPUType();                    /* CPUID命令対応か */
    if (i <= 0x400) {
         printf("This CPU does not support CPUID¥n");
         return -1;
    }
    _getCPUID(1, data);                 /* CPUID命令実行 */
    cpu=0;
    if (data[12+0] & 1) {               /* FPUチェック */
         cpu=cpu|1;     /* FPUあり */
    }
    if (data[12+2] & 0x80) {            /* MMXチェック */
         cpu=cpu|2;     /* MMXあり */
    }
    if (data[12+3] & 2) {               /* SSEチェック */
         cpu=cpu|4;     /* SSEあり */
    }

    /* ハイメモリアクセスライブラリ初期化 */
    if (_preInitHimem() != 0) {
         printf("Hi Memory LIB Initialize error¥n");
         return -2;
    }
    _maskNMI();                         /* パリティエラーNMI禁止 */


    /* sample data */
    for(j=0;j<32;j++){
         data[j]=j+1;
    }

    /* ハイメモリアクセスサンプル */
    _writeHimemByte(0x00100001,0xAA);
    c=_readHimemByte(0x00100001);

    _writeHimemWord(0x00100006,0x55aa);
    i=_readHimemWord(0x00100006);

    _writeHimemLong(0x00100008,0xff00ff00);
    l=_readHimemLong(0x00100008);

    if (cpu&1) {
         _writeHimem64(0x00100010,data);
         _readHimem64(0x00100010,data);
    }

    if (cpu&2) {
         _writeHimem64MMX(0x00100020,data);
         _readHimem64MMX(0x00100020,data);
    }

    if (cpu&4) {
         _writeHimem128(0x00100030,data);
         _readHimem128(0x00100030,data);
    }

    _readHimemBlockByte(0x00100000,data,1024);
    _writeHimemBlockByte(0x00200000,data,1024);

    _readHimemBlockWord(0x00100000,data,1024/2);
    _writeHimemBlockWord(0x00210000,data,1024/2);

    _readHimemBlockLong(0x00100000,data,1024/4);
    _writeHimemBlockLong(0x00220000,data,1024/4);

    _fillHimemByte(0x00200000,0x80000,0x12);
    _fillHimemWord(0x00300000,0x80000/2,0x3456);
    _fillHimemLong(0x00400000,0x80000/4,0x789ABCDE);

    _copyHimemByte(0x00200000,0x00280000,0x80000);
    _copyHimemWord(0x00300000,0x00380000,0x80000/2);
    _copyHimemLong(0x00400000,0x00480000,0x80000/4);

}
```

するなら，0000_000F_FFF0_0000hを指定することになります．

もちろん，キャッシュ制御レジスタが実装されているかどうかも，_getCPUType()と_getCPUID()関数で確認してください．

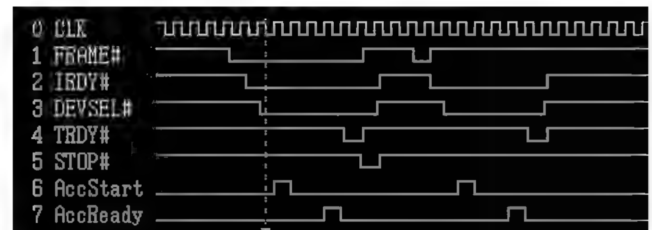
リストDにキャッシュ制御プログラムCACHE.EXEの動作を，図BにPCIメモリ空間をキャッシャブルに設定したあと，1ワードアクセスしたときのPCIバスの動作を示します．

●チップセットにより動作に違いが――あくまでハードウェアデバッグ用として使う

キャッシュ制御機能を使いPCIメモリ空間をキャッシャブル領域に設定したとしても，チップセットによっては正しく動作しない場合があるようです．筆者のテストでは，ほとんどの場合で，キャッシュフィルのためのバーストリードは発生しますが，マシンによってはライトバック動作でハングアップするものがありました．

このようなことから，ここで解説する方法によるバースト転送は，すべてのPC/AT互換機で動作可能というわけではありません．この方法でバーストリードが発生するPC/AT互換機が見つかれば，それを使って設計したPCIデバイスのバースト転送の動作確認ができますよという，あくまでハードウェアデバッグの一手段と考えるべきでしょう．この方法によるバースト転送を前提としたPCIデバイスやドライバは，推奨できるものではないと思われます．

〔図A〕64ビットアクセス時のPCIバスの動き



〔リスト C〕キャッシュ制御関数

```
int _getNumberOfCacheRangeIndex();
int _getCacheMode(unsigned int RangeIndex, unsigned char *BaseAddress,
                                              unsigned char *AddressMask, unsigned int *CacheMode, unsigned int *RangeValid);
int _setCacheMode(unsigned int RangeIndex, unsigned char *BaseAddress,
                                                unsigned char *AddressMask, unsigned int CacheMode, unsigned int RangeValid);
```

〔リスト D〕キャッシュ制御プログラムの動作

```
A:\>CAHCE

IA-32 P6 Family Cache Controller program

Index     BaseAddress Reg.          Mask Reg.         Mode
  0h   0000_0000__0000_0000h  0000_000F__E000_0000h WriteBack
  1h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  2h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  3h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  4h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  5h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  6h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  7h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable

A:\>CAHCE 7 B 80000000 00100000

IA-32 P6 Family Cache Controller program
Range Index  : 7
Cache Mode   : WriteBack
Base Address : 80000000h
Address Size : 00100000h
Cache Set success

A:\>CAHCE

IA-32 P6 Family Cache Controller program

Index     BaseAddress Reg.          Mask Reg.         Mode
  0h   0000_0000__0000_0000h  0000_000F__E000_0000h WriteBack
  1h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  2h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  3h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  4h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  5h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  6h   0000_0000__0000_0000h  0000_0000__0000_0000h Disable
  7h   0000_0000__8000_0000h  0000_000F__FFF0_0000h WriteBack
```

キャッシュ制御レジスタの状態を表示

メインメモリがライトバックに設定されている

キャッシュ制御レジスタを設定

インディスク7によりPCIメモリ8000 0000hから1Mバイトの領域がライトバックに設定された

● FPU/MMX/SSE命令使用時の注意点

今回用意したライブラリをC言語上から使うだけなら問題ありませんが，直接アセンブラでこれらの処理を記述する場合について説明します．デバッグライブラリはソースも添付しているので，関数の中で具体的にどのような命令を使っているかはそちらを参照してください．ここでは注意点を簡単に説明します．

すでに説明したように，これらの命令はCPUにより実装されているものとされていないものがあります．実行前にその機能がCPUに実装されているかどうかを確認する必要があります．

FPU命令やMMX命令についてはとくに難しいところはないでしょう．しいてあげるとするなら，EMM命令でMMX命令実行後の後始末を忘れないことです．

SSE/SSE2命令を使用するにあたってはさらに若干の注意点があります．CR4のビット9(OSFXSR)を立てないと，例外が発生する点です．このビットがクリアされていると，FXSAVE/FXRSTOR命令でSSE/SSE2のレジスタの保存復帰がされません．また(まずセットされていることはないはずだが)CR0のビット2(EM)がクリアされていることも確認します．

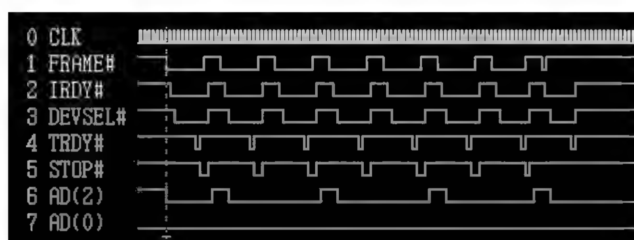
なお，SSE命令とSSE2命令は別物です．マニュアルをよくみて使い分ける必要があります．

● キャッシュ制御レジスタ変更時の注意点

キャッシュ制御関連のMTRRレジスタを制御する手順を簡単に説明します．本来なら変更前には，

1) 割り込み禁止(CLI)
2) CR4保存
3) キャッシュ禁止(CR0のCD＝1，NW＝0)

〔図B〕キャッシュフィルによるバーストリード時のPCIバスの動き



4) キャッシュフラッシュ(WBINVD)
5) ページングモード禁止(CR4 PGE＝0)
6) TLBをフラッシュ[CR3をアップデート(リードしてそのままライト)]
7) MTRRs禁止(MTRR_DEF_TYPE(2FFH)のビット11(E)＝0)

以上の処理を，変更後には，

8) MTRRs許可(MTRR_DEF_TYPE(2FFH)のビット11(E)＝1)
9) TLBをフラッシュ[CR3をアップデート(リードしてそのままライト)]
10) キャッシュフラッシュ(WBINVD)
11) キャッシュ許可(CR0のCD＝0，NW＝0)
12) CR4復帰
13) 割り込み許可

の処理をする必要があります．しかし今回のデバッグライブラリはMS-DOSのリアルモードで動作し，一部分のキャッシュ属性を変更するだけなので，

1) 割り込み禁止(CLI)
2) キャッシュフラッシュ(WBINVD)
3) 変更するMTRRのみ禁止(MTRR Physical Mask *n* Register ビット11(v)＝0)
4) MTRR Physical Base *n* Register 設定
5) MTRR Physical Mask *n* Register 設定(このレジスタにMTRRのイネーブルビットがあるのでBase Registerより後に設定する必要がある)
6) 割り込み許可

というような，かなり簡単な処理にしています．

なお，キャッシュの領域指定が重なる場合は，ライトバックとアンキャッシャブルが重なる部分は，アンキャッシャブルが優先します．それ以外の組み合わせで重なる場合は不明です．また，キャッシュ属性に指定以外(0，1，4，5，6以外)の値を設定するとハングアップなどの挙動を示します(ヘッダファイルに定義済み)．

＊　　＊

最後に，キャッシュ制御レジスタを使ってPCIメモリ空間にバースト転送を発生させる方法は，“PCIデバッグライブラリ for Win32”の作者である柏野政弘氏のアイデアをお借りしたものです．柏野氏は本業多忙のため，先にDOS版のほうに実装させていただきました．

参考文献

1)『PCIデバイス設計入門』，TECH I Vol.3，CQ出版(株)

すがわら・なおのぶ　BIOS/ファームウェアプログラマ

組み込みGUIの設計の現状とソリューション

第2回 iWin for XP Embedded による UI 開発の実際

中山 宏之

今回は，最初に iWin for XP Embedded のターゲット OS となる Windows XP Embedded について説明し，次に iWin ソリューションの成り立ちを，最後に iWin での典型的な UI 開発の手順を説明します．



Windows XP Embedded とは？

● Windows XP Embedded の特徴について

ここで，今回のターゲット OS となる Windows XP Embedded についておさらいしておきましょう．第一に，Windows XP Embedded はマイクロソフトの組み込み Windows 製品の一つです(もう一つは Windows CE.NET)．Windows XP Embedded の基本部分は，デスクトップ向け PC で動作する MUI[注1] 版の Windows XP OS そのものです．これに，組み込み向けのいくつかの追加機能と構成管理を行うためのツールやデータベース管理システムが付属しています．デスクトップ PC 向けでは MUI 版 Windows XP は特定ユーザー向けのボリュームライセンス専用製品として用意されていますが，Windows XP Embedded では MUI 機能を利用して最大 12 種類の国別リソースをもった切り替え可能 OS を生成できるようになっています．

▶コンポーネント化された OS：通常の Windows XP では，すべてのユーザーは同じ機能セット/API を使用することを想定されていますが，Windows XP Embedded ではコンポーネント単位でモジュールを削除できます．Windows XP Embedded には，このようなコンポーネントの管理を行うためのツール「コンポーネントデザイナ」,「コンポーネントデータベースマネージャ」が付属しています．

▶Windows XP のすべての機能が利用可能：必要であればクライアント OS である Windows XP のすべての機能が利用可能です．また Windows XP 向けのサードパーティ製アプリケーションやデバイスドライバも(もちろんバイナリコードのまま)そのまま動作させられます．Windows XP の幅広いドライバ対応は，そのまま広範なハードウェアが利用可能なことを意味します．

▶Embedded 向けとして追加された独自機能：Embedded 向け機能として，OS イメージファイル(SDI)を作成する機能や，これを PXE[注2] と組み合わせて実現するリモートブート機能などがサポートされています．

▶開発ツール：上述したコンポーネント管理のためのツールや，OS を開発するための「ターゲットアナライザ」,「ターゲットデザイナ」などが付属しています．また，アプリケーション開発が必要な場合には，通常の Visual Studio .NET や他のデスクトップ向け開発ツールを利用することが可能です．

▶安いライセンス料：最後に，これはマイクロソフトの資料には出てきませんが，通常のデスクトップ向け Windows XP(やそのボリュームライセンスプラン)と比べてライセンス料が安いということがあげられます．実際には使用するコンポーネントの種類と数にもよるのですが，これまで通常の Windows 製品を使って組み込み製品を作っているところでは，Windows XP Embedded にするだけで費用の節約になります．とはいっても，Windows CE.NET や他の無料 OS と比べるとやはりそれなりの価格にはなります．

逆に，Windows XP Embedded を使用するにあたって必ず考えておかなければならないことがあります．

▶ウイルス，ワーム対策：ネットワークに接続するような機器では，少なくとも出荷時の時点で問題が発生しないよう，必要な QFE[注3] を必ず当てておくべきです．そして必ずファイアウォール機能を動作させ，使用するポート以外を閉じておくようにします．また，どうしても OS を更新しなければならないときのため，ユーザーアップデートの手段を準備しておく必要があります．

● Windows XP Embedded に必要な UI ソリューション

Windows Embedded OS も，もちろん GUI 画面をもたないヘッドレス機器用の OS として使用することが可能ですが，や

注1：Multilingual User Interface. 基本となるプログラムコードに選択可能な各国語用リソースを追加したもの．コントロールパネルの設定で，OS 標準アプリケーションの使用言語を切り替えられる．

注2：Preboot Execution Environment，インテルの開発したネットワークブートの方法．`http://www.intel.com/labs/manage/wfm/wfmspecs.htm` を参照のこと．

注3：Quick Fix Engineering，いわゆるパッチのこと．`http://www.microsoft.com/windows/embedded/xp/downloads/default.asp` を参照．

はり GUI が必要な機器に使用してこそ，その本領を発揮すると思います．とはいえ，ここに問題点が一つ存在します．Windows XP Embedded がデスクトップ Windows XP と同じ機能をもつというのは一つの特徴ではあるのですが，例のエクスプローラー，スタートメニュー，タスクバーを使った UI シェルを組み込み製品にもそのまま利用したいというユーザーはどれくらいいるのでしょうか？　残念ながら標準の Windows XP Embedded では，エクスプローラーシェル（とコマンドプロンプト）以外の UI シェルは用意されていません．このような状況では，ユーザーは次のいずれかの手法をとるように思われます．

① ターゲット機で動作する UI アプリケーションを一つに限定する
② 自分でシェルアプリケーション（簡単なラウンチャなど）を開発する

シェルとして動作するプログラムの存在しない①では，唯一動作しているこの UI プロセスが異常終了したときには，機器全体を再起動するしか復帰する方法がありません．Windows XP の標準エクスプローラーシェルの場合には同時に監視プロセスが動いていて，これが異常終了したときには自動的にプロセスを再起動するようになっています．これに対し②では，自分であらゆる場合を想定して好きなようにシェルアプリケーションを構築できますが，もちろん開発費，開発期間，開発ノウハウなどの面で問題が発生します．

じつは iWin for XP Embedded には，次のような Windows の UI シェルとしての機能が搭載されており，このような組み込みの応用にも対応できるように作られています．

- 外部アプリケーションの起動，切り替え，強制終了
- すべてのトップレベルウィンドウの管理
- 残りメモリが設定量以下になったときに自動ハイバネーションに入る
- デスクトップウィンドウの提供（エクスプローラーの置き換え可能）
- システム起動時の自動アプリケーション起動
- システムのシャットダウン
- シェルアプリの自動再起動（通常は `IWin.exe` を再起動プロセスに指定する）



iWin for XP Embedded とは？

● ローカルブラウザベース UI について

前回の記事で，iWin ソリューションとは「ローカルブラウザベース UI を構築するための開発キット」である，という説明をしました．ローカルブラウザベース UI とは，簡単にいえば，全画面表示されたブラウザの上に HTML で記述して構築する UI です．ルータやサーバなどのヘッドレス（表示画面をもたない）機器でも，同様に HTML 記述の UI が使用されます．こちらも HTML ファイルによって UI 画面を作成するのですが，（Web）サーバベース UI といって，UI を実現するためのしくみが異なります．

一般に，ローカルブラウザベース UI は次の特徴をもちます．これらは iWin でも同様です．

▶ グラフィカルな UI を簡単に作成できる：さまざまなグラフィックファイルを UI 画面に取り込むことが（ファイル名を記述するだけで）できます．これは画像表示のためのプログラムや，少なくとも画像表示用コンポーネントが必要になる他の UI 構築手法に対する大きな利点です．また，グラフィックを表示させたまま画面をデザインできる Web 開発ツールや，Web 用データがそのまま利用可能という利点もあります．DHTML を使えば，外部コンポーネントを使用せずに簡単な動きを表現することが可能です．

▶ 拡張性が高い：Web ブラウザのもつローカルスクリプト機能を用いて，（ボタンを押したときなどの）さまざまなイベントの動作を記述できます．また Windows Embedded OS では機能拡張用 ActiveX コンポーネントを利用し，UI にいろいろな機能拡張を追加できます．たとえば Macromedia Flash や Java アプレットも（やろうと思えば）UI 画面に採り入れられます．

▶ 変更が容易：HTML ファイル（やその中のスクリプト）はテキストファイルとして記述されています．そのためコンパイルすることなく使用ファイルや動作内容を変更できます．デザイナさんとプログラマの連携もスムーズに行えます．

一方，単にブラウザベース UI としただけでは，次の欠点が残ってしまいます．

▶ ブラウザ画面の表示内容を十分な速度で切り替えるのが難しい：組み込み向けはパソコン用と比べて速度の遅い CPU が使われる傾向にあります．このような状況で Web 向けの画面表示は，テキストによる表示や単純なグラフィック表示に比べて描画に時間がかかることになります．また，ある UI 画面から別の UI 画面への表示切り替え速度はブラウザのキャッシュに依存する部分が出てきてしまいます．

▶ 長方形以外の形のウィンドウを作るのが苦手：もとはブラウザコントロールなので，楕円形で浮かんでいるような表示ウィンドウを作る，などは難しいものがあります．

▶ 使用するリソースがこれまでの単一アプリケーションより多くなる：ブラウザベース UI ではグラフィカルな画面を簡単に作成できる反面，どうしても（グラフィックファイルなどに）使用するメモリやストレージ容量が大きくなる傾向にあります．

これらの欠点を緩和するため iWin では，

▶（外部プログラムまで含んだ）すべての（トップレベル）ウィンドウのコントロールとイベント処理を記述
▶ 複数のブラウザウィンドウを使用して画面を分割する．あるいは使用するブラウザウィンドウを起動時に（裏で）読み込んでおく

などの手法が利用できます．また使用リソースの問題についても，最近は CPU スピードが速くなり，使用可能なメモリ容量



も増えているため，このようなリッチUIを搭載した組み込み機器も実現可能になってきました．

● iWin for XP Embeddedのおもな機能

iWin for XP Embeddedに含まれる機能やモジュールを項目としてあげると，次のようになります．

▶Windows XP Embeddedですぐに使えるUIソリューション：iWin for XP EmbeddedをWindows XP Embedded開発環境にインストールすると，付属の.SLDファイルを利用してiWinのコンポーネントが自動的にデータベースにインポートされます．そのため，Target Designerを利用してすぐにiWinコンポーネントを使用したOSイメージを作成できます．

〔表1〕iWinで使用するコンポーネント

モジュール名	ProgID注	機　能
iwin.exe	−	iWinプロセスの起動
IWinSystem.dll	BSQUARE.IwinSystem	システムコンポーネント
IWinShell.dll	BSQUARE.IwinShell	シェル機能
IWinBrowser.dll	BSQUARE.IwinBrowser	ブラウザ機能
BSQKeyboard.dll	BSQUARE.KeyboardObject	キーボードイベント処理
BSQFileSystemObj.dll	BSQAURE.FileSystemObject	ファイルシステム操作
BSQRegistry.dll	BSQUARE.RegDatabase	レジストリ操作
BSQWinsock.dll	BSQUARE.Tcp BSQUARE.Udp BSQUARE.Daemon	ソケット通信機能
BSQDial.dll	BSQUARE.Dialer	RAS/VPNダイヤル

注：スクリプト内でオブジェクトを生成するときに使う文字列．iWin内ではvar objkBD=iWin System, Createobjct"BSQVARE. Keyboard Object"; のように生成する

▶ローカルブラウザベースのUI構築：iWinではおもにWindows XP Embeddedに含まれるIEブラウザコンポーネントを利用してUI画面をユーザーに提示します．そのため，iWinにはIEコントロールを拡張するiWin用のブラウザコンポーネントが含まれます．このブラウザコンポーネントは「ローカルスクリプト」を利用してブラウザクラスごとのイベント記述が可能になっています．

▶グローバルスクリプトのサポート：iWinの起動スクリプトとして利用可能な「グローバルスクリプト」がサポートされます．このスクリプトを利用してiWinシステムで使用するコンポーネントの開始を記述できます．ちなみにiWinで使用するスクリプトエンジンは，Windows XP Embedded組み込みのJScriptエンジン（の拡張）です．

▶シェル機能のサポート：通常のエクスプローラーシェルがない環境でiWinをシェルとして使用するための（前述したような）シェル機能がコンポーネントにより提供されます．この機能により，スタートメニューと同等のタスク管理を（やろうと思えば）実現できます．

▶便利なiWin付属コンポーネント：iWinソリューションでは，前述したコンポーネント以外にも任意のActiveXコンポーネントを利用できます．Windows XP Embeddedシステムに標準で含まれるActiveXコントロールもありますが，それ以外にもiWinはファイルシステム操作，レジストリ操作，ダイヤルアップ接続など，便利に使用できる追加コンポーネントを含んでいます．追加コンポーネントを含めたすべてのiWinコンポーネントのリストを表1に示します．

▶UIサンプルDLL：iWinで構築するUIはもともとはHTMLファイル，JSファイルと画像ファイルにより構成され，一般に多数のファイルを使用することになります．そのため，ひとと おりUIの開発が終わった段階で使用するすべてのデータファイルを格納した単一のリソースDLLを作成することがあります．このようなUIサンプルの単一DLLとしてiWinSample1.DLL，iWinSample2.DLL，そしてiWinSample3.DLLの三つのDLLが提供されます．

▶ドキュメント：iWinソリューションの考え方，開発方法などを説明したユーザーズマニュアルと，付属コンポーネントの提供するオブジェクトごとの使用可能プロパティ，メソッド，イベントを記述したリファレンスマニュアルが提供されます．マニュアルには便利な“How to …”ノウハウを記述した内容も含まれます．

● iWinの有効な使用法について

iWinを使用するにあたって，いくつかヒントとなることを紹介します．

▶キーボードオブジェクトの利用：iWin付属のキーボードオブジェクトを利用すると，いわゆるホットキーイベントを実装できます．これはWindowsの任意の仮想キーコード（VKEY）を受け取るよう指定できるため，たとえばデバッグ時用にいずれかのファンクションキーを利用するような実装が簡単にできます．一方，専用のハードウェアによるキーを用意し，これを押したときに通常使わない仮想キーコードをイベントキューに入れるAPIを呼び出すことでも，このしくみを有効に活用できます．

▶画面分割，画面切り替えのテクニック：通常のWebサイトで画面の左や上にメニューを置くときには，ブラウザ内のフレームやスタイルシートで実現する方法がよく用いられます．一方，iWinでは画面を複数のブラウザで分割し，一方にメニュー専用のHTMLを配置することでメニューを実現できます．iWinでは生成されたブラウザのリストを管理しているため，メニューブラウザから簡単に別ブラウザの要素にアクセスできます．これにより，メニュー部分の書き換えにともなう全画面の再描画を抑止できます（図1）．

よく使用するブラウザオブジェクトをバックグラウンドに表示しておくこともできます．通常のWebサイトでの画面切り替えはおもにHTMLファイルの読み込みにより常にその場で行われ，しばしば全画面の書き換えになります．これに対して

iWinでは起動時にあらかじめ複数のブラウザオブジェクトを生成すると同時にHTMLファイルを読み込んでおき，メニュー選択によって後からそのウィンドウを瞬時に前面に出せます．

▶window.alertの代わりにIWinSystem.MessageBoxを使う：通常のWin32プログラムでは，しばしばMessageBox APIを利用して簡単な情報表示を行います．JScriptにもMessageBox APIを呼び出す`window.alert`関数が用意されていますが，いわゆるWindowsスタイルのメッセージボックスが表示されるため，iWinで使用するには違和感があります．そのかわり`IWinSystem.MessageBox`メソッドを使用することで，iWinスタイルのメッセージボックスを表示できます．

▶バックグラウンドブラウザを利用してダイアログを実現：iWinの通常のUI画面はいわゆるボーダーエリアを使用せずにブラウザで画面を分割することでできます．ところが，ブラウザクラスを生成し，BackgroundURLプロパティにボーダーエリアとして使用するグラフィックやクローズボタンなどのUIエレメントを含んだHTMLファイルを指定することで，ダイアログのような表示を行うブラウザクラスを定義できます．BackgroundURLで使用するHTMLファイルでは任意のグラフィックが使用できるため，いわゆるスキンのような働きをもたせることができます．

▶XMLファイル，データベースの利用：iWinでそれほど多量ではない（せいぜい数十件くらいの）データ扱うときには，XMLファイルとしてデータを用意しておくと便利です．ご存知のようにXMLファイルはスタイルシートを利用することで，表として簡単にブラウザ内で表示することができます．また，自前でHTML要素を組み立てることも可能です．こうして生成したHTML要素をテーブル要素のinnerHTMLに設定することで，iWin画面の一部にXMLファイル由来の情報を表示します（リスト1）．もう少し量のあるデータを使用するときは，ADO[注4]をローカルHTML内で使用します．Windows XP Embedded組み込みのMDBデータベースや，（ネットワーク経由で）外部のSQLデータベースを利用することももちろん可能です．

iWinソリューションの開発

ここでは，iWinソリューションを実際に開発する場合の手順を順に説明します．

● iWinによるUI開発の準備

▶開発用ワークステーションの準備

Windows XP Embedded開発を行うためには，Windows 2000 SP3以降，またはWindows XPをインストールした開発機を用意します．この上でMSDE（開発用データベースエンジン）を稼動させる必要があるため，256Mバイト以上のメモリが必要です．グループ開発を行う場合，MSDEの代わりにSQLサーバを用意すれば，メンバ間で開発用データベースエンジンを共有できます．

▶ターゲット機の仕様を決める：Windows XP Embeddedのターゲット機は基本的に，Windows XPが動作可能なx86ベースのハードウェアとなります．必要に応じてネットワーク機能やローカルハードディスクの有無を決定します．場合によってはコンパクトフラッシュからOSを起動したり，PXEによるネットワークブートも使用できます．

▶ユーザーインターフェースデバイスの決定：Windows XP Embedded自体はGUIを使用しないヘッドレス構成も可能ですが，iWinを使うということで当然グラフィック画面を利用したインターフェースということになります．ここでは，

- 画面の大きさ，画面の解像度は，CRTか液晶か？
- マウス使用か，タッチパネル使用か，
- フルキーボードを使用するか？　それともいくつかの専用ボタンのみか？
- 音声入出力を利用するか？

〔図1〕複数のブラウザによる画面分割



〔リスト1〕XMLファイルの例

```
<Applications>
   <Programs>
      <Program>
         <Name>Freecell</Name>
         <Path>C:¥WINNT¥System32¥freecell.exe</Path>
         <Icon>images/programs_freecell.gif</Icon>
      </Program>
      <Program>
         <Name>Solitaire</Name>
         <Path> C:¥WINNT¥System32¥sol.exe</Path>
         <Icon>images/programs_solitare.gif</Icon>
      </Program>
      <Program>
         <Name>Command Prompt</Name>
         <Path>C:¥WINNT¥System32¥cmd.exe</Path>
         <Icon>images/system.gif</Icon>
      </Program>
   </Programs>
</Applications>
```

注4：Active Database Objectの略．Windows XP Embedded組み込みのMDBデータベース機能をActiveX経由で操作できる．



●そのほか，特徴的なデバイスを使用するか?(認証デバイス，ICカードなど)

などの選択を行います．

● コーディング前段階

▶UI仕様の決定：実際のUI開発作業に入る前に，あらかじめUI仕様を決めておき，事前にもれがないか，使いにくくはないかなどを検討します．UIの使いやすさは，ある単一の画面デザインがどうかということよりも，使用シーンを想定し，うまくユーザーを誘導できるかどうかにかかっています．UI仕様レベルで操作の流れを検討しておくことにより，必要なメニュー項目や画面数を明確にします．これらの作業の結果，次の項目が決定します．

- 想定されるユーザー
- UIで使用するデバイス
  ディスプレイ機構，入力/ポインティングデバイス，表示や音声によるフィードバックの有無，セキュリティデバイス(ICカードなど)の有無
- UI操作のシナリオ
- 使用する(用意する)画面のリスト

▶UIタスクの分析：UIで必要となる画面を決定するための手法として，次の二つの方法が考えられます．

●機能リストの分析：あらかじめ機器で実現するべき機能が決まっている場合，それを選択するためのUIが必要になることがわかります．たとえば①メール，②Webブラウズ，③インスタントメッセンジャーの三つの機能の実現が要求されているとき，少なくともこれら三つのシーン画面とこれらを切り替えるためのメニューを作成することになります．

●タスク中心の分析：ユーザーが成すべきタスクを考えることによって，機器の実現するべき機能を決定する方法もあります．もしこの機器がインターネットを介したオンライン発注専用端末であるとすると，たとえば①サーバとの接続，②発注するアイテムと数の入力，③発注者情報の入力，④発注情報の印刷の四つのシーンが考えられます．もし印刷機能が必要ということになれば，それに関連して「使用可能プリンタの選定」が必要になり，「印刷に使用するプリンタ」の設定画面が必要になることも考えられます．以下に，iWinでよく使用されるタスクの例をあげます．

- ログインスクリーン，オンラインユーザー登録
- Webブラウズ
- メール
- メディアプレーヤ
- ローカルファイル，メモリカードのブラウズ
- 設定画面(コントロールパネル的なもの)
- ソフトウェア的またはハードウェア的なパニックボタンの使用

▶ 使用するコンポーネントの検討：iWin内で通常のHTMLやJScriptで実現できない機能を提供するために，Windows XP Embedded組み込みコンポーネント，iWinの付属コンポーネント，自作，あるいはサードパーティのActiveXコンポーネントといった，各種ActiveXコンポーネントを使用できます．

▶ 画面デザインの検討：UI仕様中で各UI画面がもつべき機能が決まると，今度はそれぞれのUI画面内のどの部分(ボタンなど)に機能を割り振るか，という画面デザイン作業に入ります．一方，一つの機器の中で明らかにほかと異なる調子をもつ画面が出てくると，ユーザーは混乱します．そのため，①全体のデザイン(調子)の統一，②各シーンに必要な独自性をもたせる，という二つのバランスをとりながら画面を構成していきます．この段階でもできれば何度もレビューして，機能の割り付け方に無理がないか，デザインにこりすぎていないか，などの検討を加えます．

実際の画面グラフィックの作成はデザイナに発注することになるかもしれません．iWinの画面はHTMLファイル＋グラフィックファイルで構成されるので，デザイナとのデータのやり取りがスムーズで，画面イメージの検討がブラウザで簡単にできます．

● iWinソリューションのコーディング

必要な画面イメージファイルがそろったところで，必要な機能を組み込むためのコーディングに入ります．iWinで使用するスクリプトはすべて，JScriptで記述します．

▶ グローバルスクリプトの作成：グローバルスクリプトは，iWinを起動したときに`IWinSystem`オブジェクト内で読み込まれます．グローバルスクリプトには`IWinSystem_Startup`，`IWinSystem_Shutdown`，`IWinSystem_OnError`の三つの必須ファンクションとオプションのイベント処理ファンクションを記述します．この中で重要なのはもちろん`IWinSystem_Startup`関数で，iWinスタート時に最初に実行されます．次のような処理を記述します．

- グローバルオブジェクト(変数)の生成
- 必要なブラウザクラスオブジェクトの生成
- ブラウザクラスよりブラウザのインスタンスを生成(初期画面，バックグラウンド画面の生成)
- (必要に応じて)iWinShellの起動
- (必要に応じて)キーボードオブジェクトの生成(パニックボタンの実装)

▶ ローカルスクリプトの作成：iWinではUIを構成するブラウザオブジェクトは，①ブラウザクラスオブジェクトの生成，②ブラウザインスタンスの生成の2ステップで行われます．つまり，同じ属性をもつブラウザをいくつでも簡単に生成できます．ブラウザクラスを使用するもう一つの理由が，ローカルスクリプトの存在です．ローカルスクリプトは，ブラウザクラスごとに必要な処理を一箇所にまとめて記述するという役割をもっています．ローカルスクリプトの指定はそれ自体オプションですが，これを使用することによりブラウザクラスごとの`IWinBrowser_Startup`，`IWinBrowser_Shutdown`，`IWinBrowser_OnError`の三つのオプションファンクション

とイベント処理ファンクション，その他任意のファンクションを記述できます．

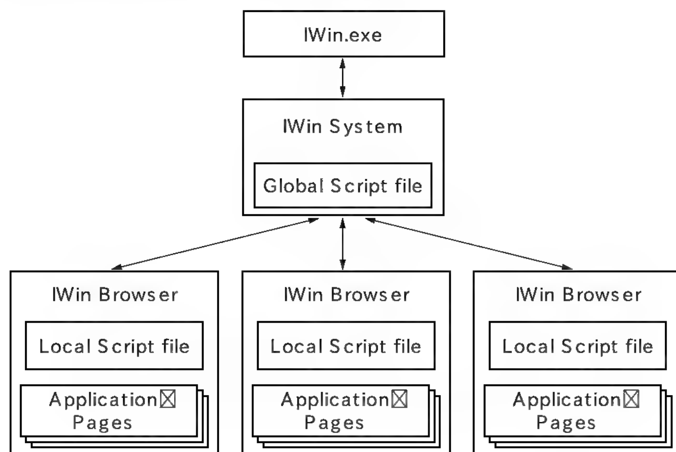
▶ アプリケーションページ（HTML）作成：ブラウザクラスから生成された各ブラウザオブジェクトは「アプリケーションページ」のHTMLによって実際に画面表示されます．アプリケーションページのもとは画面デザイン時に作成したHTMLファイルを利用しますが，それだけではなくiWin特有のオブジェクトアクセスによって機能を埋め込むこともできます．アプリケーションページは普通のWebページのように複数のHTMLファイルで構成でき，個々のHTMLファイル内にも，ページ内でローカルなJScriptイベントなどを記述できます．ブラウザインスタンス間の連携は，各ブラウザクラスで生成したオブジェクトやグローバルスクリプトで生成したオブジェクトへアクセスすることで達成できます．グローバルスクリプト，ローカルスクリプト，アプリケーションページと各オブジェクトの関係を図2に示します．

▶ 処理内容の記述：各スクリプトやアプリケーションページで記述されるiWin内での実質的な処理は，①JScriptを直接利用した情報処理，②（ブラウザがネイティブにもっている機能を利用しての）インターネットへのアクセス，③ActiveXオブジェクトやJava Appletの利用，④外部のWin32プログラムの起動のうちいずれかによってなされます．

たとえば，①のJScript記述によって簡単な電卓機能などを実装できます．②のインターネットアクセス機能を利用すれば，いわゆるWebサーバとのHTTPのPOSTやGET，あるいはXMLウェブサービスを利用できます．③のActiveXを利用すると，Macromedia Fashやその他ActiveXで実現された機能を利用でき，これらActiveX中からもさらにネットワークにアクセスできます．

最後に④のiWinShellの提供するシェル機能により，通常のWin32アプリケーションを起動できます．Win32アプリケーションは，起動後は基本的にiWinのコントロール下から外れますが，iWinShellの機能により外部プログラムのウィンドウ

〔図2〕オブジェクトとスクリプトの関係



の生成や終了をiWinのイベントとして検出できます．

▶ インクルードファイルの使用：iWinスクリプトではあらかじめ用意された各種Constファイルが利用可能です．たとえばグローバルスクリプト内では，

```
IWinSystem.Include( "IWinSystemConst.js" );
```

ローカルスクリプト内では，

```
IWinBrowser.Include("IWinBrowserConst.js");
```

のような形で利用します．

これらのインクルードファイルのパスはiWinプロセスのカレントディレクトリからの相対パスになり，実行ファイルのようなサーチパスのしくみは存在しないので注意が必要です．場合によっては「`CONST_PATH + "IWinSystemConst.js"`」のような絶対パスの形で記述します．

● iWinの起動方法

iWinを起動するにはグローバルスクリプトの絶対パスをパラメータとして`iwin.exe`を実行します．インクルードファイルを相対パスで記述しているときは，現在のディレクトリをインクルードファイルのあるフォルダへ移動したあと`iwin.exe`を起動します．

● iWinソリューションのデバッグ

iWinで利用可能なデバッグ手法を紹介します．

▶ ログファイルの使用：IWinSystemとIWinBrowserはそれぞれ独立したログファイルにメッセージを出力する機能をもっています．コンポーネント自体のエラー出力と，`IWinSystem.DebugMessage`，`IWinBrowser.DebugMessage`の両関数によるスクリプトからのエラー出力が可能です．

▶ メッセージボックスの使用：`IWinSystem.MessageBox`関数により，メッセージボックスをUI画面に表示できます．

▶ エラーハンドラの使用：グローバルスクリプトやローカルスクリプトで発生したエラーイベントは，それぞれのスクリプト内の`IWinSystem_OnError`，または`IWinBrowser_OnError`関数でキャッチできます．これらのハンドラ内でエラーイベントの種類や発生箇所を判別しつつエラー処理を記述します．一方，`IWinSystem.RaiseError`，`IWinBrowser.RaiseError`でプログラム的にエラーイベントを発生させることもできます．

▶ 予防的プログラミング：スクリプトプログラミングでは，メソッド呼び出しの返り値がオブジェクトであることが多いですが，このような局面エラーチェックをせずにすぐに帰ってきたオブジェクトを使ってしまうようなコードを書くと，たまたまエラーが発生し，nullが返ってきた後メソッド呼び出ししようとしたところでスクリプトエラーになります．もちろんエラーイベントはエラーハンドラでキャッチできますが，ここは正しくnullが返ってきたかどうかの判断を行うべきです．

▶ デバッガの使用：じつはiWin for XP Embeddedの各コンポーネントは，通常のデスクトップWindowsに登録（regsvr32）することにより，ターゲット機とほぼ同じように動作させるこ

とができます．そして，iWinソリューションをデスクトップWindowsで動かすことの大きな利点は，デバッガが使用できることです．iWinではスクリプトエンジンとしてWindowsに付属するWSH[注5]をそのまま使用しているため，Active Script Debugger Interfaceを使用する任意のスクリプトデバッガでデバッグできます．このようなデバッガには，

- Windows Script Debugger 1.0[注6]
- Visual InterDev 6.0に付属するスクリプトデバッガ
- FrontPage2002などに付属するスクリプトデバッガ（図3）

などがあります．

これらのスクリプトデバッガで実際にiWinスクリプトをデバッグするには，次のようにします．

① スクリプトエラー時にデバッガが呼び出されるようにレジストリを設定します．

```
[HKEY_CURRENT_USER¥Software¥Microsoft¥
                    Windows Script¥Settings]
"JITDebug"=dword:00000001
```

を設定します．

② 初期化スクリプト中に“debugger;”ステートメントを記述したあとiWinを起動し，いったんデバッガを起動した後，デバッガで追跡したい個所にブレークポイントを設定します．

以上の手順により，スクリプトエラーが発生した箇所か，あるいはブレークポイントを設定した箇所でデバッガによるデバッグが可能になります．

● リソースDLLの作成

iWinソリューションの開発が一段落したら，関連のHTMLファイル，JSファイル，グラフィックファイルをひとまとめにしてリソースDLLにしてしまうことをおすすめします．リソースDLLを使用することにより，関連する多数のデータファイルを一つのファイルにまとめられるので管理が楽になり，バージョンアップのときの処理も簡単になります．

リソースDLLを作成するには，少なくともリソースコンパイラが必要です．Visual StudioなどのIDEツールがあれば，次の方法でより簡単にDLL化できます．

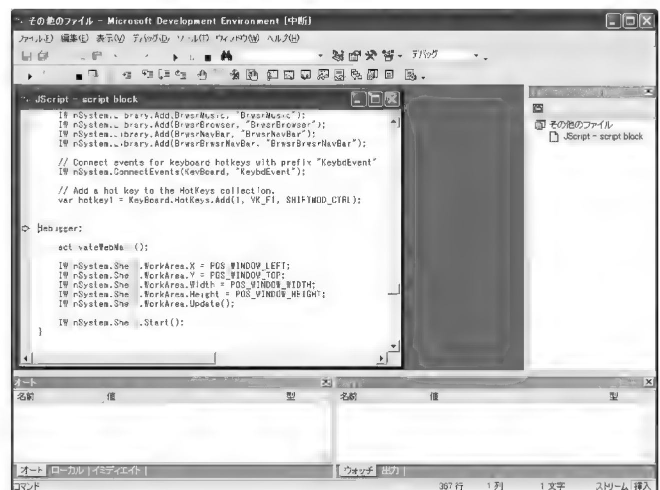
① HTMLファイルとJSファイルは定義済みリソースタイプHTMLでリソース中にインポートします．

② グラフィックファイルはカスタムリソースタイプ“IMAGES”でインポートします．

③ 最後に，グローバルスクリプトをカスタムリソースタイプ“SCRIPT”でインポートします．

たとえばHTMLファイル“sample.htm”，イメージファイル“image.gif”，スタートアップスクリプト“start.js”をこの方法でres.dllに格納したとすると，HTMLファイルは“res://res.dll/sample.htm”，イメージファイルは相対パスで“images/image.gif”，グローバルスクリプトは“res://res.dll/SCRIPT/startr.js”でアクセス可能です[注7]．

〔図3〕FrontPage2002のスクリプトデバッガ



〔表2〕レジストリキー

キー	種　類	説　明
[HKLM¥SOFTWARE¥BSQUARE¥iWin]		
DefaultExecutable	文字列	iwin.exeの絶対パスを指定
DefaultCommandLine	文字列	コマンドライン引き数を指定
[HKLM¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion¥Winlogon]		
Shell	文字列	iwin.exeの絶対パスを指定

● Windows XP Embeddedイメージの作成

最後に，iWinソリューションで使用しているiWinコンポーネントとリソースDLLをWindows XP Embeddedイメージに組み込みます．iWinをログオン後のシェルとして使用するときは，表2のレジストリキーを使用します．

以上でiWinの組み込みが完了しました．次回は，iWinで使用しているWindowsのスクリプトエンジンの解説と，iWinでも使用可能な独自ActiveXコントロールの簡単な作成法を説明します．

なかやま・ひろゆき　ビースクウエア（株）

注5：Windows Scripting Hostの略．

注6：マイクロソフトのホームページからダウンロード可能．http://www.microsoft.com/downloads/details.aspx?FamilyID=2f465be0-94fd-4569-b3c4-dffdf19ccd99&DisplayLang=en（英語版），http://www.microsoft.com/downloads/details.aspx?FamilyID=e606e71f-ba7f-471e-a57d-f2216d81ec3d&DisplayLang=ja（日本語版）

注7：この方法はIEの各種ダイアログでも利用されている．ためしにIEのアドレスバーに“res://shdoclc.dll/ABOUT.DLG”と入れてみてほしい．

IrDAを使った機器を手軽に開発するための⊠

「IrFront H8S Trial Kit」の詳細⊠

渡辺 一弘/岩田 吉弘/荻野 直晃

はじめに

「IrFront H8S Trial Kit」は，ルネサス製H8Sプロセッサ，シリアルインターフェース，IrDAトランシーバを搭載した名刺大のボードに，ACCESSのIrFront IrDA プロトコルスタックをプリインストールしたものである．2003年12月号に掲載しだ「IrFront H8S Trial Kit」の概要』では，その簡単な紹介を行った．そこで今回は，ソフトウェア編とハードウェア編にわけて，より詳細な解説を行う．

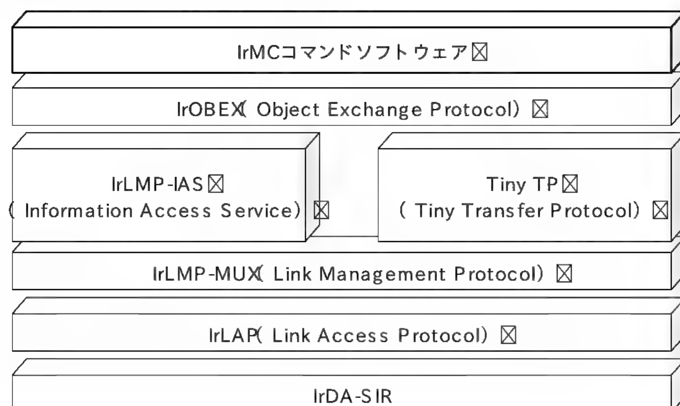
ソフトウェア編

本編では，IrFront H8S Trial Kit(以下Trial Kit)に付属するソフトウェアについて説明する．このTrial Kitには，すでにH8S用にコンパイル済みのIrMCコマンドソフトウェアがROMに書き込まれている．このTrial Kitで，①IrDA 評価(携帯との接続性，スピード感，障害対応性など)，②携帯電話からのユーザーデータ(電話帳，メール，ブックマーク，写真データなど)の受信評価，③iアプリを使った赤外線通信の評価などを行える．なお，このTrial Kit上のソフトウェアを開発することはできない．

IrFront H8S Trial Kitのソフトウェア構成

本ソフトウェアは，ACCESS社IrDA プロトコル開発キットであるIrFront v2.0 SDK 基本パッケージとIrMCパッケージをH8S用に移植したものである．また，携帯との赤外線通信の評価を簡単できるようにするためにシリアルからのIrMCコマンドを受け付け，IrMC/OBEX のデータを変換する機能をもつ．図1 の細枠が，IrFront v2.0 SDK のパッケージに含まれている部分であり，太枠のIrMCコマンドソフトウェアがこのキットのために開発されたソフトウェアである．

〔図1〕Trial Kitのソフトウェア構造



IrMCコマンドソフトウェアは，次の機能をもつ．

① シリアルポートからIrDA・IrMCのコマンドの受付，メッセージ表示機能
② IrDA 各種パラメータを変更機能
(a) ディスカバリパラメータの変更(SLOT，インターバル)
(b) IrDA QoS(一部固定)
(c) メッセージのON/OFF
③ IrOBEXヘッダ設定機能
(a) ヘッダ設定送信機能
(b) ヘッダ受信表示機能
(c) OBEX 通信ログ表示ON/OFF
(d) 3種類のOBEXヘッダをサポート(表1)
④ iアプリサンプル連携機能

Trial Kitの内容および必要機材

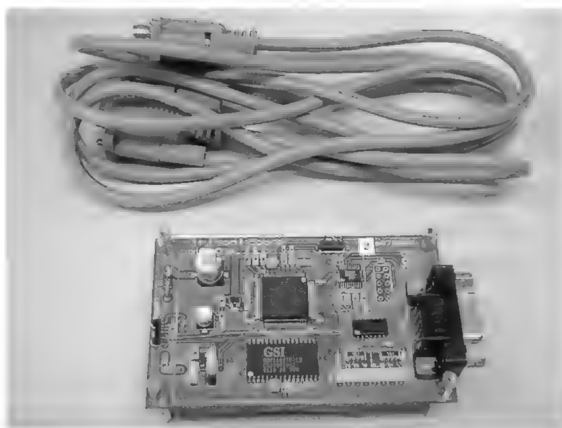
本Trial Kit(写真1)には，以下のものが同梱されている．

① IrFront H8S評価ボード(H8S/2148)
② RS-232-Cケーブル(ストレート): IrFront H8S評価ボードの方ですでに回線をクロス接続しているので，PCと接続する場合はストレート接続となる．
③ CD-ROM
1) ドキュメント: IrFront H8S Trial Kitハードウェアマニュアル(含む回路図)とIrFront H8S Trial Kit操作マニュアル

〔表1〕対応OBEXヘッダ

対応OBEXヘッダ	設定制限など
NAME	ASCII 128バイト以内
TYPE	ASCII 128バイト以内
BODY・END BODY	30Kバイト以内(30Kバイト以上の受信は，メモリに取得しないが，通信は切らない)

〔写真1〕Trial Kitの内容



2) ソフトウェア：iアプリサンプルソースコードとIrFront H8S Trial Kitソフトウェアバイナリ

本キットを動作させるためには，次のものを用意する必要がある．

① シリアルポートのあるPC
② 単3電池×3個（または安定化電源と配線ケーブル）
③ ターミナルソフト（Windows標準搭載のハイパーターミナルなど）
④ iアプリ開発環境：iアプリを開発するためには，NTTドコモのホームページより，開発ツール一式をダウンロードする必要がある．また，できあがったJavaプログラムを携帯にダウンロード可能なサイトが必要である（詳しくは，http://www.nttdocomo.co.jp/p_s/imode/を参照のこと）．

PCとの接続とTrial Kitの設定

● PCとの接続

IrFront H8S評価ボードとPCは，付属のRS-232-Cケーブルを使い接続する（図2）．PCの通信ポートの設定は，表2のようになっている．

● Trial Kitの設定

Trial Kitは，電源が供給されると，図3のようなメッセージを出力する．

“IrFront>”は，プロンプトである．IrMCコマンド入力待ちとなっている．じつはこの状態で，すでにIrMCコマンドがサーバモードでスタートしている．携帯のユーザーデータを受信可能な状態である．したがって，ターミナルソフトがなくとも赤外線通信可能だが，その結果を表示するためにも，用意していただきたい．

このTrial Kitでできることは？

このTrial Kitでは，①携帯ユーザーデータのダウンロード，②iアプリ連携機能，③PC以外の装置との接続機能を実験できる．以降，順に説明する．

● 携帯ユーザーデータのダウンロード

携帯のユーザーデータをTrial Kitにダウンロードできる（図4）．携帯のユーザーデータをダウンロードすることで，どのようなデータが送信されているか内容を知ることができる．携帯のユーザーデータは，アドレス帳，カレンダ，メール，Bookmark（URL），写真データなどである．なお，iアプリ版の写真データは仕様がメーカー独自であるため，取得できない．

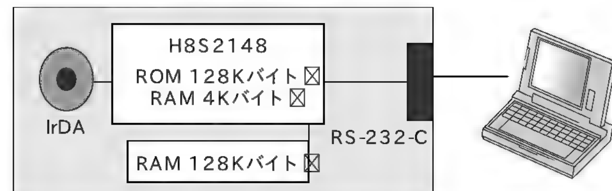
● iアプリ連携機能

本Trial Kitには，赤外線通信を行うiアプリサンプルが付属している．iアプリサンプルは，三つある（詳細は後述）．付属のiアプリサンプルはソースで提供されており，これを使うためには，iアプリ開発環境と作成したiアプリを置くためのサーバ（自分のWebページサイトなど）が必要である（図5）．

〔表2〕PCの通信ポートの設定

ポートの設定項目	設定内容
ビット・秒	115200bps
データビット	8
パリティ	なし
ストップビット	1
フロー制御	なし

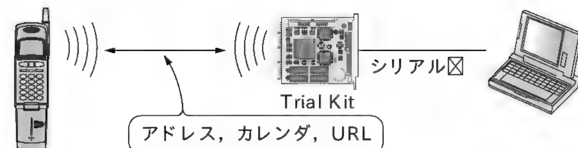
〔図2〕PCとの接続



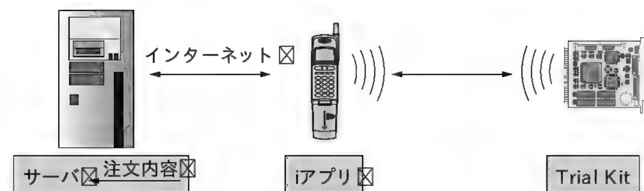
〔図3〕メッセージ

```
***************************************************
***   IrFront H8S Trial Kit v1.0               ***
***                  Rel 1.0  2003/10/10       ***
***     Copyright(C) 2003  ACCESS CO.,LTD.     ***
***************************************************
IrFront>
```

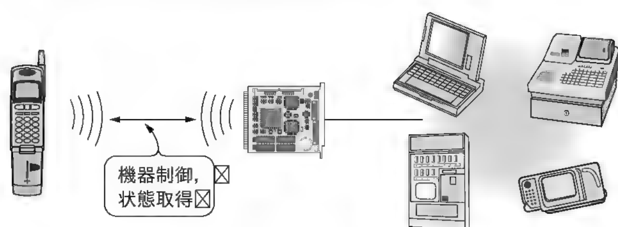
〔図4〕携帯ユーザーデータのダウンロード



〔図5〕iアプリ連携機能



〔図6〕PC以外の装置との接続確認



① 会場エントリｉアプリ
② OBEXTEST送受信ｉアプリ
③ おみくじｉアプリ

● PC以外の装置との接続確認

Trial Kitは，ターミナルからIrMCコマンドを使って，OBEXのデータをやり取りできる．これを応用して，赤外線をもたない装置（シリアルポートは必要）が，携帯とやり取りすることが可能となる（図6）．

Trial Kit IrMCコマンドとそのメッセージ

本Trial KitボードのおもなIrMCコマンド一覧を表3に，おもなIrMCコマンドのメッセージを表4に示す．

〔表3〕おもなIrMCコマンド一覧

			コマンド	入力例・備考
IrDA基本設定	ディスカバリ	実行時間	IRDA DIS TIM {NNN}	IRDA DIS TIM 30
		間隔	IRDA DIS INT {NNN}	IRDA DIS INT 3
		スロット	IRDA DIS SLT {NN}	IRDA DIS SLT 6
		表示	IRAD DIS	現在の設定を表示
	QOS	スピード	IRDA QOS SPD	最大スピードをセット
		DATASIZE	IRDA QOS DAT	最大データサイズをセット
		DISCTIME	IRDA QOS DIS	切断時間をセット
		表示	IRDA QOS	現在の自局QoSを表示
IrMC OBEX設定	開始	CLINET	IRMC STA CLI	CLINETモードをスタート
		SERVER	IRMC STA SER	SERVERモードをスタート
	終了		IRMC STO	停止
	HEADER SET	NAME	OBEX SET NAME {NAME}	NAMEをセットする
		TYPE	OBEX SET TYPE {TYPE}	TYPEをセットする
		BODY	OBEX SET BODY {BODY}	BODYをセットする
DEBUG	OBEX送受信DUMP		OBEX DUMP ON	送受信NNNバイトDUMP表示
			OBEX DUMP OFF	DUMP停止
	メッセージ		IRDA MSG [ON¦OFF]	メッセージのON/OFF

〔表4〕おもなIrMCコマンドのメッセージ

メッセージ	意　味
*** DISCOVERY [OK¦NG]	NGはTIMEOUTが発生
*** IRLAP [OK¦NG] [SPD=9600,DAT=1024,WIN=1, BOF=0,MIN=0.5,MAX=500,DIS=8]	OKの時は，QoSを表示 SPD=接続スピード DAT=相手DATA SIZE WIN=相手WINDOW SIZE BOF=追加すべきBOFS MIN=最小ターンアラウンドタイム MAX=最大ターンアラウンドタイム DIS=切断スレッシュホールドタイム
*** IRIAS [OK¦NG]	IASの結果を表示（クライアントの場合）
*** IRTTP [OK¦NG]	TTP接続の結果を表示
*** OBEX SND CONNECT	CLIENT 動作 OBEX CONNECT を送信
*** OBEX RCV RESPONSE　[OK¦NG(XX)]	CLIENT 動作RESPONSEを受信
*** OBEX RCV GET COMMAND	SERVER動作GET COMMANDを受信
*** OBEX SND RESPONSE OK	SERVER動作 RESPONSEを送信
*** OBEX RCV NAME {NAME}	NAMEを受信．その内容を表示
*** OBEX RCV TYPE {TYPE}	TYPEを受信．その内容を表示
*** OBEX RCV BODY {BODY}	BODYを受信．その内容を表示
*** OBEX RCV ABORT	OBEX ABORTを受信
*** IRMC END [OK¦NG] {NNNN}	NNNN　msディスカバリからのOBEX DISCONNECTまたはNGまでの経過時間（単位ms）

携帯のユーザーデータを受信する

Trial Kitの電源をONした直後から，IrMCサーバがスタートしている．この状態では，ターミナルソフトを使ってIrMCコマンドを入力することなく，赤外線データの送受信を行うことができる．図7は，携帯からアドレス帳を送信したときのターミナルソフトの表示例である．DISCOVERY OK（サーバモードの場合応答をレスポンス）したときから内部タイマがスタートし，OBEX DISCONNECTレスポンス返すまでの時間を計測している．これにより，約20種類以上もある携帯ごとの赤外線通信スループットの計測が可能である．カメラ付き携帯の写真データをH8SクラスのCPUで受信可能か否か評価することもできる．

ｉアプリとの連携

本Trial Kitを使って，ｉアプリと通信を行う．本Trial Kitには，三つのｉアプリを提供している．

● 会場エントリｉアプリ

ｉアプリ上で，ユーザー情報（名前とエントリID）を登録する．この情報をTrial Kitに送信するサンプルである（図8）．使用するOBEX HEADERは，NAME，TYPE，BODYの3種類である．NAME HEADERにより，会場エントリｉアプリであることを特定できる名前をつける．TYPE HEADERは，BODYのMIMEタイプを指定するものであり，特定アプリ間との通信でありBODYのフォーマットが決まっているなら，とくに使用する必要はない．しかし，サンプルなので，あえて，TYPE HEADERを使うこととする．BODYには，名前とエントリIDを区別するタグを付ける．

● OBEXTEST ｉアプリ

ｉアプリ上にて，OBEXの送受信

〔図7〕携帯からアドレス帳を送信したときのターミナルソフトの表示例

```
IrFront＞
*** DISCOVERY OK
*** IRLAP OK
*** [SPD=38400,DAT=512,WIN=1,BOF=0,MIN=10,MAX=500,DIS=8]
*** IRTTP OK

*** OBEX RCV CONNECT COMMAND
*** OBEX SND RESPONSE SUCCESS(0x A0)
*** OBEX RCV PUT COMMAND
*** OBEX RCV NAME [pb.vcf]
*** OBEX RCV BODY
BEGIN:VCARD
VERSION:2.1
N;CHARSET=SHIFT_JIS:アクセス 太郎;;;;
SOUND;X-IRMC-N;CHARSET=SHIFT_JIS:ｱｸｾｽ ﾀﾛｳ;;;;
TEL;VOICE:090XXXXXXXX
EMAIL;INTERNET:XXXXXXXXXXX@docomo.ne.jp
X-CLASS:PUBLIC
END:VCARD
*** OBEX SND RESPONSE SUCCESS(0x A0)
*** OBEX RCV DISCONNECT COMMAND
*** OBEX SND RESPONSE SUCCESS(0x A0)
*** IRDA END OK  [260]
*** FCS ERROR 0
*** RESEND    0
```

<解説>
①IrLAPの接続時の相手QoSを表示
　SPEED＝38400bps DATASIZE＝512バイト ADD BOFs＝0バイト
　最小ターンアラウンドタイム＝10ms
　最大ターンアラウンドタイム＝500ms
　切断スレッシュホールドタイム＝8秒
②IrTTP接続完了後，OBEX CONNECT/OBEX PUT/OBEX DISCONNET パケットを受信
③OBEX PUTは，NAMEヘッダ，BODYヘッダを含んでいた
　（a）NAMEヘッダは，pd.vcf
　（b）BODYヘッダには，BEGIN：VCARDからEND：VCARD
④ディスカバリ受信から，OBEX DISCONNETを返信するまで，約860ms
⑤本送受信におけるFCSエラー（FCSERROR），再送（RESEND）発生数は0

をテストすることができる（図9）．iアプリからの設定項目は表5のとおりである．

▶iアプリの制限による本OBEXTESTの機能の制限

iアプリを使って赤外線通信を行うと，「赤外線通信を行いますか」とメッセージが表示され，かつ，通信中「赤外線通信中」のメッセージが表示され続ける．このメッセージは，IrDAの接続中，IrDAの切断まで表示され続ける．このため，OBEX CONNECT/PUT/GET/DISCONNECTの動作をユーザーにより選択することは不可能となっている．このためOBEXTESTアプリは，通信途中にコマンド選択できないため，GET/PUTのいずれかを事前に選択して，1コマンドのみ送受信する仕様となっている．

● おみくじiアプリ

本Trial Kitの応用として，おみくじアプリケーションを搭載している．このアプリケーションは，Trial Kitの電源投入後動作しているので，ぜひ購入直後の運を占っていただきたい．

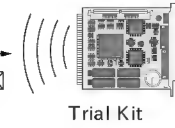
● vTriggerサンプル

vTriggerは，外部機器から携帯のiアプリを起動させる機能である．この機能を使うことで，ユーザーによるiアプリの起動選択を行わず起動が可能となる．vTrigger起動には，ADF

〔図8〕会場エントリiアプリ

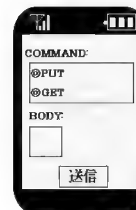
使用するOBEX HEADER	内　容
NAME	ENTRY.DAT
TYPE	text/plain
BODY	X-ENTRY-NAME:アクセス太郎 X-ENTRY-ID:12345

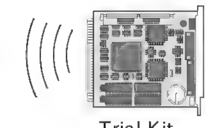


ユーザー情報を送信

Trial Kit

```
*** OBEX CONNECT OK
*** OBEX PUT OK
*** OBEX RCV NAME [ENTRY.DAT]
*** OBEX RCV TYPE [text/plain]
*** OBEX RCV BODY
X-ENTRY-NAME:ÉAÉNÉZÉXêæòY
X-ENTRY-ID:12345
*** OBEX DISCONECT OK
```

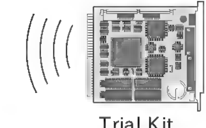
〔図9〕OBEXTEST iアプリ




Trial Kit

Trial Kitは，事前に設定された内容に基づき，応答を返す．




Trial Kit

〔表5〕iアプリからの設定項目

設定項目	内容/制限など
動作モード	CLIENT/SERVER
CLIENT時 COMMAND	PUT or GET
SERVER時 RESPONSE	SUCCESS / BAD REQUEST / NOT IMPLEMENT
NAME	128バイト ASCII
TYPE	128バイト ASCII
BODY	256バイト ASCII

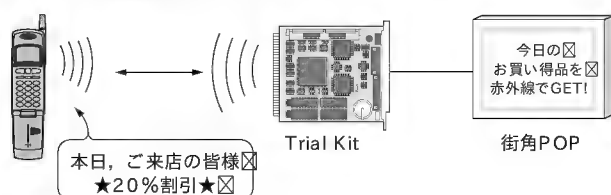
〔表6〕ADF設定

ADF設定	内　容
AllowPushBy	Irda:sample

設定を表6のように設定し，ソフトウェアをダウンロードし直す必要がある．これによりvTriggerによる起動が行えるようになる．vTriggerによるiアプリの起動のため，携帯を赤外線受信状態にする必要がある（この操作は，機種に依存する）．

（N504iの例）「メニューボタン」→「ツールBOX」→「赤外線」

〔図10〕お店情報を送るシステム



PC以外の装置との接続連携

本Trial KitのIrMCコマンドを活用することで，赤外線をもたない装置でも携帯とやり取りすることが可能となる．図10は，街角のPOPにTrial Kitをつけ，そのお店のURL，クーポン，お店情報を送るシステムである．これにvTriggerを使えば，そのお店のアプリケーションを間違いなく起動することができ，顧客はサービスを取得できる．これは，とても大事なことである．赤外線によるサービスが増えてきているため，目的のアプリケーション確実に起動させるこのvTriggerの機能は，iアプリの操作が難しいユーザーの手助けとなるだろう．赤外線をかざすユーザーにサービスを配信できることは，とても魅力的なことである．

ただし，一つ問題がある．vTriggerを使うためには，携帯を赤外線受信状態にしなければならない．これが各社，機種ごとに操作が異なる．そこで提案だが，iモードボタンならぬ，IrDAボタンをつけたらどうだろうか．となれば，街角に赤外線POPが増えること間違いないと思うのだが．

ソフトウェア編のまとめ

IrFront H8S Trial Kitの開発は，H8SクラスのCPUでどのくらいの処理ができ，かつ携帯との通信の使用感がどのくらいなのかを評価したいと希望する顧客の声からスタートした．開発当初は，外付けRAM上に受信データを格納しつつ，FCSの計算などを行ったため，38400bpsで通信したときでさえ受信取りこぼしが発生した．H8SのRAMは4Kバイトである．割り込み処理中のデータのやり取りをこのRAMにおいやり，SIRドライバを多少チューニングして，115200bpsのデータを受信できるまでに至った．115200bpsの赤外線通信を行いながら，かつ別タスクを行うのは難しいと思われるが，赤外線送受信に専念するしくみがとれるのであれば，H8Sで十分機能すると思われる．省電力機能をもつH8Sを使って携帯と情報をやり取りできるソリューションは，家電，街角，どこにでも，そしていつでも情報を提供できるサービスの普及に大いに役立つだろう．

ハードウェア編

「IrFront評価ボード」は，ルネサステクノロジ製の16ビットマイクロプロセッサH8S/2148，RS-232-Cインターフェース，ローム製IrDA Ver.1.2トランシーバRPM872，外部メモリ1Mバイトを搭載した名刺大のサイズで，低コストで容易にユーザーシステムの短期開発を可能にする．

IrFront評価ボードに使用したH8S/2148はルネサスオリジナルアーキテクチャを採用したH8S/2000CPUを核に，システム構成に必要な周辺機能を集積した1チップマイクロコンピュータである．H8S/2000CPUは，内部32ビット構成で，16ビット×16本の汎用レジスタと高速動作を指向した簡潔で最適化された命令セットを備えており，16Mバイトのリニアなアドレス空間を扱える．また，H8/300およびH8/300H CPUの命令に対し，オブジェクトレベルで上位互換を保っているので，H8/300，H8/300L，H8/300Hの各シリーズから容易に移行できる．H8S/2148はROM 128Kバイト，RAM 4Kバイトを内蔵している．内蔵ROMはフラッシュメモリ(F-ZTAT)で，仕様流動性の高い応用機器，量産初期から本格量産の各状況に応じた迅速かつ柔軟な対応が可能である．本LSIはシリアルインターフェースを3チャネル内蔵しており，うち1チャネルをIrDAインターフェースとして使用できる．また，外部バスインターフェースを使用し，外部メモリへの接続も容易に実現できる．“IrFront Trial Kit”は電池駆動，低コストで実現させる必要があり，低消費電力でIrDAインターフェースを内蔵した廉価なH8S/2148を採用した．

回路図を図11に，機能仕様を表7に示す．

〔表7〕IrFront評価ボードの機能仕様

項　目	仕　様
CPU	H8S/2148　HD64F2148A 内蔵ROM　128Kバイト 内蔵RAM　4Kバイト
SRAM	GSI製　1Mバイト(128Kバイト×8)
システムクロック	動作周波数：7.37MHz セラミックレゾネータ 村田製作所製　CSTCE7M37
ターゲットへのシリアル接続	H8S/2148内蔵チャネル1(CN3)
デバッグ用ボードシリアル接続	H8S/2148内蔵チャネル0(CN2)
IrDAトランシーバシリアル	H8S/2148内蔵チャネル2(U3)
ボーレート	115200bps
電源	安定化電源(3.3V)　もしくは単3電池3個

低コスト化のために

本ボードは低コストで実現させるため，

① リチウムイオン電池ではなく，単3乾電池3個とツェナダイオードと抵抗で動作電圧に必要な3.3Vを作り出している
② 基準クロックは高価な水晶発信子の代わりにセラロック(村田製作所製CSTCE7M37G 7.37MHz)を使用している



〔図11〕IrFront H8S Trial Kit の回路図

●「IrFront H8S Trial Kit」読者プレゼントのお知らせ

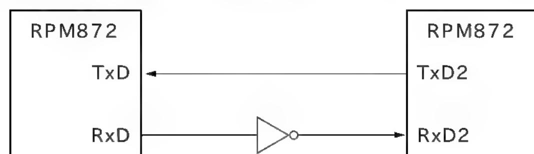
本稿と2003年12月号記事「IrFront H8S Trial Kit」の概要』で解説してきたIrFront H8S Trial Kitを，開発元のご厚意により1台，読者プレゼントします．

ご希望の方は，本誌読者アンケートはがきに必要事項をご記入のうえ，「希望の品名」に「IrFront Kit」と記して，2003年12月26日(必着)までにご応募ください．なお当選者の発表は，発送をもってかえさせていただきます．

IrFrontは，IrDAの仕様に準拠し，ACCESSが独自開発した情報家電･専用機器向けのコンパクトな赤外線通信プロトコルスタックです．iモード対応携帯電話をはじめ，多くの機器への搭載実績があります．IrCOMM，IrTran-Pを標準サポートし，赤外線決済を実現するIrFMへの対応のほか，IrMC/IrOBEX/OBEXなどのオプションのを用意しています．504iシリーズよりサポートされているiアプリ連携機能「vTrigger」，MCPCのオブジェクト交換インプリメンテーションガイドラインの仕様に準拠した「IrMC Level4」に対応しています．

またTrial Kitは，ACCESSのWebページ，または代理店から，3万円(税別)で購入することも可能です．ACCESSでは，このキットを使ったiアプリ開発のセミナーを12月16日(火)に開催する予定です．詳細はWebページをご参照ください(http://www.access.co.jp/)．

〔図12〕H8S/2148-IrDA接続例



③ LEDはローム製低消費LED SML211を使用し，H8S/2148-LED間をバッファロジックを介さず，ダイレクトに接続している

④ 基板構成は両面基板．SRAMへのアクセスを8ビット幅にすることで，基板上の配線数を減らしている．H8S/2148は16ビットの外部データバス幅をもっているが，外部データバス幅を8ビットに設定した場合は上位8ビットのみが有効となる

基準クロックの選定方法

IrDA Ver.1.2は，データ転送速度115kbps/通信距離30cmを実現する必要がある．H8S/2148で115kbpsの調歩同期式通信を実現するために，使用クロックを選定した．H8S/2148のシリアル転送速度を決める，SCIモジュールのビットレートレジスタBRRの値は，次の式で表される．

$$N = \frac{\phi}{8\boxtimes 2\boxtimes^{\boxtimes -1\boxtimes} \times B} \times 10\boxtimes -1\boxtimes$$

B：ビットレート(bps)

N：ボーレートジェネレータのBBRの設定値

ϕ：動作周波数(MHz)

n：ボーレートジェネレータ入力クロック(n=0)

H8S/2148の3V時の最大周波数が10MHzであることから，上記式より115200bpsを実現するためには，N=1として計算するとϕの値は7.37MHzとなる．

H8S IrDAインターフェース

H8S/2148は1チャネルのIrDAインターフェースを内蔵しており，複雑な回路を組まなくても容易にIrDAトランシーバと接続できる(図12)．キーボードコンパレータコントロールレジスタのIrDAイネーブルビットを設定することで，SCIチャネル2のTxD2，RxD2の信号はIrDA規格に準拠した波形のエンコード/デコードを行う．これを赤外線送受信トランシーバ/レシーバと接続することで，IrDA規格に準拠した赤外線送受信を実現できる．本ボードはIrDA規格バージョン1.2を実現しており，通信は115200bpsの転送レートで通信を開始する．その後の転送レートの変更はH8S/2148のレジスタを設定することにより，ソフトウェアで変更できる．

送受信および接続方法

IrDAインターフェースからの出力信号はSCIからのUARTフレームをIRフレームに変換し出力する．シリアルデータが0のときビット幅の3/16のHighパルスの信号に変換される．IrDAトランシーバRPM872からH8S/2148への入力信号はシリアルデータが0のときビット幅の3/16のLowパルスの信号で入力されるため，本ボードにはRxD2の前段にルネサステクノロジ製超小型ユニロジックインバータを搭載している．

本ボードにはIrDA以外に2チャネルのシリアルを使用できる．SCI1チャンネルにはRS-232Cドライバを介してD-Sub9ピンコネクタとつながっており，ユーザーPCとシリアルクロスケーブルで接続できる．SCI0チャネルはスルーホールに接続されている．ユーザーボードと組み合わせてのシステム構築が容易に構成できるようになっている．また，汎用ポートをスルーホールに出しているため，ユーザーのシステム拡張も可能である．

おわりに

現在，携帯電話のIrDAを用いて各種サービスが展開されようとしている．本ボードを使うことで，ユーザーはUARTインターフェースを使って容易に携帯電話と通信できるシステムを構築できる．

参考文献･URL
1)(株)ルネサス テクノロジ，『H8S/2148シリーズ ハードウェアマニュアル』
2) http://www.renesas.com/

わたなべ･かずひろ (株)ACCESS
いわた･よしひろ/おぎの･なおあき (株)日立超LSIシステムズ

フリーソフトウェア徹底活用講座

第13回

続々・GCC2.95から追加変更のあったオプションの補足と検証

岸 哲夫

今回は，前回(2003年11月号)に引き続き，GCC2.95から追加変更のあったオプションの補足と検証を行う．「コード生成規約に対するオプション」を扱う．

(筆者)

さて，GCC3.3はだいぶ普及してきたようです．まだRPMパッケージになっているものは見かけませんが，10月に発売されたTurboLinux 10 Desktopはカーネル2.6を採用し，GCCは3.3が含まれています．話題のLindowsに対抗する意味合いで「turbopkg」が発展し「クイック・イン」と呼ばれるものになったようです．Lindowsの「Click-N-Run Warehouse」に比肩する使用感かどうかは，機会があったら検証してみます．β版が発表され雑誌の付録に付く予定もあるようですが，いろいろ問題があるようです．Turbolinux公式サイト(`http://www.turbolinux.co.jp/`)を参考にしてください．

コード生成規約に対するオプションの補足

次のオプションは，追加されたものです．

● -fbounds-check

このオプションは，GCCのFORTRAN言語のフロントエンドであるg77や，同じくJava言語のフロントエンドであるGCJで使用するオプションです．ここで詳しい説明はしません．このオプションを付加すると，加算，減算，乗算で発生する符号付きのオーバフローに対するトラップを生成します．もちろん処理速度は落ちますが，デバッグ中に使うと便利です．コンパイルと実行の結果を次に示します．

```
$ gcc test189.c -o test189
$ ./test189
20
0
-1524072448
$ gcc test189.c -o test189 -ftrapv
$ ./test189
20
0
アボートしました
$
```

このようにオーバフローして異常な値を検出したら，異常終了します．たいていの場合，オーバフローした値はメモリを破壊したりして，簡単には原因がわからないバグを生み出します．内部的に異常な数値を抱えたままプログラムが走ってしまうことを考えると，このオプションをデバッグ中につけておけば気持ちも晴れやか(?!)になるでしょう．ソースと生成されたコードをリスト1～リスト3に示します．

このように加算，減算，乗算の命令が発生するごとにそれぞれ`__addvsi3`，`__subvsi3`，`__mulvsi3`を呼び出しています．

● -fnon-call-exceptions

`throw`例外をトラップすることを可能にするコードを生成します．これが，どこでも存在するとは限らないプラットホームに特有のランタイムサポートを要求することに注意してください．さらに，それは例外(つまりメモリ参照，浮動小数点命令)をトラップすることを単に可能にします．それは，`SIGALRM`のようなシグナルをトラップする処理とは関係ありません．C++

〔リスト1〕加算，減算，乗算に対する符号付きのオーバフローに対するトラップを生成する例(test189.c)

```
/*
 *加算，減算，乗算に対する符号付きのオーバフローに対するトラップを生成
 */
#include <stdio.h>
int main()
{
    int  data1;
    int  data2;
    int  res;
    data1     =    10;
    data2     =    10;
    res  =    data1     +    data2;
    printf("%d¥n",res);
    res  =    data1     -    data2;
    printf("%d¥n",res);
//実際にオーバフローさせる処理

    data1     =    1000000;
    data2     =    8000000;
    res  =    data1     *    data2;
    printf("%d¥n",res);
    return    0;
}
```

特有の機能なので，ここでは触れません．このオプションをつけてもエラーにはなりませんが，単独では意味のないコードが付加されるだけです．ただし C++ で書かれた例外処理とリンクしている場合は，このオプションを有効にする必要があります(リスト 4)．

● -funwind-tables

C++ の例外処理を有効にします．詳しくは後で解説します．通常，例外処理を必要としない C 言語では，デフォルトでこのオプションは無効となります．ただし C++ で書かれた例外処理とリンクしている場合は，このオプションを有効にする必要があります．例外処理を記述していないソースのコンパイルにこのオプションをつけてもエラーにはなりませんが，単独では意味のないコードが付加されるだけです(リスト 5)．

● -fasynchronous-unwind-tables

もしターゲットマシンがサポートしているなら，warf2フォーマット形式のデバッグ情報ファイルを使います．デバッガやガベージコレクションで発生する非同期イベントによってスタックをアンワインドするために使用できます．

アンワインドとは，

```
main()
 ↓  func1()
 ↓          func2()
 ↓                  func3()
```

のように関数 `func3` を実行中に `func1` に戻る必要が出た場合，`func2` に戻らないためスタックを操作する必要ができます．それがアンワインドです．「スタックの巻き戻し」とも呼ばれてい

〔リスト 2〕オプションなしで生成されたアセンブラソース(test189.s)

```
    .file    "test189.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $24, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $10, -4(%ebp)
    movl $10, -8(%ebp)
    movl -8(%ebp), %eax
    addl -4(%ebp), %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl -8(%ebp), %edx
    movl -4(%ebp), %eax
    subl %edx, %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $1000000, -4(%ebp)
    movl $8000000, -8(%ebp)
    movl -4(%ebp), %eax
    imull     -8(%ebp), %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

〔リスト 3〕オプション付きで生成されたアセンブラソース(test189a.s)

```
    .file    "test189.c"
.globl __addvsi3
.globl __subvsi3
.globl __mulvsi3
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $40, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $10, -4(%ebp)
    movl $10, -8(%ebp)
    movl -8(%ebp), %eax
    movl %eax, 4(%esp)
    movl -4(%ebp), %eax
    movl %eax, (%esp)
    call __addvsi3
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl -8(%ebp), %eax
    movl %eax, 4(%esp)
    movl -4(%ebp), %eax
    movl %eax, (%esp)
    call __subvsi3
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $1000000, -4(%ebp)
    movl $8000000, -8(%ebp)
    movl -8(%ebp), %eax
    movl %eax, 4(%esp)
    movl -4(%ebp), %eax
    movl %eax, (%esp)
    call __mulvsi3
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

〔リスト 4〕test189.c から生成されたアセンブラソース(test189b.s)

```
    .file    "test190.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
.LFB3:
    pushl     %ebp
.LCFI0:
    movl %esp, %ebp
.LCFI1:
    subl $24, %esp
.LCFI2:
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $10, -4(%ebp)
    movl $10, -8(%ebp)
    movl -8(%ebp), %eax
    addl -4(%ebp), %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl -8(%ebp), %edx
    movl -4(%ebp), %eax
    subl %edx, %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $1000000, -4(%ebp)
    movl $8000000, -8(%ebp)
    movl -4(%ebp), %eax
    imull     -8(%ebp), %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
.LFE3:
    .size     main, .-main
    .section
.eh_frame,"a",@progbits
.Lframe1:
    .long     .LECIE1-.LSCIE1
.LSCIE1:
    .long     0x0
    .byte     0x1
    .string   ""
    .uleb128 0x1
    .sleb128 -4
    .byte     0x8
    .byte     0xc
    .uleb128 0x4
    .uleb128 0x4
    .byte     0x88
    .uleb128 0x1
    .align 4
.LECIE1:
.LSFDE1:
    .long     .LEFDE1-.LASFDE1
.LASFDE1:
    .long     .LASFDE1-.Lframe1
    .long     .LFB3
    .long     .LFE3-.LFB3
    .byte     0x4
    .long     .LCFI0-.LFB3
    .byte     0xe
    .uleb128 0x8
    .byte     0x85
    .uleb128 0x2
    .byte     0x4
    .long     .LCFI1-.LCFI0
    .byte     0xd
    .uleb128 0x5
    .align 4
.LEFDE1:
    .ident    "GCC: (GNU) 3.3"
```

〔リスト5〕test189.cから生成されたアセンブラソース(test191.s)

```
    .file     "test191.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
.LFB3:
    pushl     %ebp
.LCFI0:
    movl %esp, %ebp
.LCFI1:
    subl $24, %esp
.LCFI2:
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $10, -4(%ebp)
    movl $10, -8(%ebp)
    movl -8(%ebp), %eax
    addl -4(%ebp), %eax
    movl %eax, -12(%ebp)
    movl -12(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
.LFE3:
    .size     main, .-main
    .section
.eh_frame,"a",@progbits
.Lframe1:
    .long     .LECIE1-.LSCIE1
.LSCIE1:
    .long     0x0
    .byte     0x1
    .string   ""
    .uleb128 0x1
    .sleb128 -4
    .byte     0x8
    .byte     0xc
    .uleb128 0x4
    .uleb128 0x4
    .byte     0x88
    .uleb128 0x1
    .align 4
.LECIE1:
.LSFDE1:
    .long     .LEFDE1-.LASFDE1
.LASFDE1:
    .long     .LASFDE1-
.Lframe1
    .long     .LFB3
    .long     .LFE3-.LFB3
    .byte     0x4
    .long     .LCFI0-.LFB3
    .byte     0xe
    .uleb128 0x8
    .byte     0x85
    .uleb128 0x2
    .byte     0x4
    .long     .LCFI1-.LCFI0
    .byte     0xd
    .uleb128 0x5
    .align 4
.LEFDE1:
    .ident    "GCC: (GNU) 3.3"
```

ます．C++ではこのような単純事例だけではありません．

● -fshort-wchar

Windowsとの互換上wchar_t をunsigned shortとして2バイトで定義するか，GCCでのデフォルトのとおりsigned long integerとして4バイトで定義するかを決定します．WINEを使う場合にはWindowsとの互換を取らないと動作しないと思います．ちなみにCygwinのgccではunsigned shortとして2バイトで定義されています．コンパイルと実行結果を次に示します．

```
$ gcc test192.c -o test192
$ gcc test192.c -fshort-wchar -o test192a
$ ./test192
4
$ ./test192a
2
$
```

ソースと生成されたコードをリスト6～リスト8に示します．

● -fstack-limit-register=reg/-fstack-limit-symbol=sym/-fno-stack-limit

指定した値やシンボルのアドレスを越えてスタックが確保されないことを保証するためのコードを生成します．スタックが指定アドレスを超えて確保される場合，シグナルがレイズされます．ターゲットマシンの多くは，スタックが境界を越える前にシグナルがレイズされます．したがって，特別の警戒をとらずに，シグナルを認識することは可能です(リスト9)．

```
$ gcc test193.c -fstack-limit-register=
                                    cx -S
```

上記のようにコンパイルした場合，生成されるコードはリスト10のようになります．

● -ftls-model=model

このオプションは，スレッド固有のデータを格納するための場所を割り当てる手段を指定するものです．-fpicオプションを指定しなければデフォルトは"initial-exec"，指定してい

〔リスト6〕wchar_tのサイズをオプションで変更する例(test192.c)

```
/*
 *wchar_tのサイズ
 */
#include <stdio.h>
int main()
{
    printf("%d¥n",sizeof(wchar_t));
    return    0;
}
```

〔リスト7〕オプションなしで生成されたアセンブラソース(test192.s)

```
    .file     "test192.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $4, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

〔リスト8〕オプション付きで生成されたアセンブラソース(test192a.s)

```
    .file     "test192.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $8, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl $2, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

〔リスト9〕スタックの制限をオプションで変更する例(test193.c)

```
/*
 *スタックの制限をオプションで変更する例
 */
#include <stdio.h>
int main()
{
    int  tbl[5];
    int  ix;
    for(ix=0;ix<20;ix++)
    {
        tbl[ix]  =   ix;
        printf("%d¥n",ix);
    }
    return    0;
}
```

れば“global-dynamic”がデフォルトです．後はlocal-dynamic，local-execが指定できます．tlsはThread Local Storageの略です．

● -finstrument-functions

このオプションは，関数の入口と出口にプロファイル用の呼び出しを生成します．関数の中に入った直後と関数から出る直前に，次の名称のプロファイル用の関数が呼び出されます．引き数は，対象の関数のアドレスとその呼び出し箇所です．プロトタイプは次のようになります．

```
void __cyg_profile_func_enter(
        void *this_fn, void *call_site);
void __cyg_profile_func_exit(
        void *this_fn, void *call_site);
```

ソースと生成されたコードをリスト11～リスト13に示します．

オプションを付けると，__cyg_profile_func_enterと__cyg_profile_func_exitがそれぞれ関数の最初と最後に呼ばれていることがわかります．この関数を利用すれば，トレースを行うこともできます．

さて，変更されたオプションはたくさんあるので次回以降にまわします．表1に，コード生成規約に対するオプションをまとめます．次回は「コード生成規約に対するオプション」の補足,「最適化オプション」の補足を行う予定です．

きし・てつお

〔リスト11〕-finstrument-functionsを指定する例(test194.c)

```
/*
 *-finstrument-functions????
 */
#include <stdio.h>
int      test1();
int      test2();
int      test3();
int main()
{
    printf("%d¥n",test1());
    printf("%d¥n",test2());
    printf("%d¥n",test3());
    return    0;
}
int      test1()
{
    return    100;
}
int      test2()
{
    return    200;
}
int      test3()
{
    return    300;
}
```

〔リスト10〕オプション付きで生成されたアセンブラソース(test193a.s)

```
    .file     "test193.c"
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    subl $72, %esp
    andl $-16, %esp
    movl $0, -48(%ebp)
    movl %esp, %eax
    subl %ecx, %eax
    cmpl -48(%ebp), %eax
    jae  .L2
    int  $5
.L2:
    subl -48(%ebp), %esp
    movl $0, -44(%ebp)
.L3:
    cmpl $19, -44(%ebp)
    jle  .L6
    jmp  .L4
.L6:
    movl -44(%ebp), %edx
    movl -44(%ebp), %eax
    movl %eax, -40(%ebp,%edx,4)
    movl -44(%ebp), %eax
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    leal -44(%ebp), %eax
    incl (%eax)
    jmp  .L3
.L4:
    movl $0, %eax
    leave
    ret
    .size     main, .-main
    .ident    "GCC: (GNU) 3.3"
```

〔表1〕コード生成規約に対するオプションのまとめ

バージョン2.95	バージョン3.3	
fcall-saved-reg	fcall-saved-reg	
fcall-used-reg	fcall-used-reg	
fexceptions	fexceptions	
ffixed-reg	ffixed-reg	
	fnon-call-exceptions	追加
	funwind-tables	追加
	fasynchronous-unwind-tables	追加
finhibit-size-directive	finhibit-size-directive	
fcheck-memory-usage		削除
fprefixfunction-name		削除
	finstrument-functions	追加
fno-common	fno-common	
fno-ident	fno-ident	
fno-gnu-linker	fno-gnu-linker	
fpcc-struct-return	fpcc-struct-return	
fpic	fpic	
fPIC	fPIC	
freg-struct-return	freg-struct-return	
fshared-data	fshared-data	

バージョン2.95	バージョン3.3	
fshort-enums	fshort-enums	
fshort-double	fshort-double	
	fshort-wchar	追加
fvolatile	fvolatile	
fvolatile-global	fvolatile-global	
fvolatile-static	fvolatile-static	
fverbose-asm	fverbose-asm	
fpack-struct	fpack-struct	
fstack-check	fstack-check	
	fstack-limit-register=reg	追加
	fstack-limit-symbol=sym	追加
fargument-alias	fargument-alias	
fargument-noalias	fargument-noalias	
fargument-noalias-global	fargument-noalias-global	
fleading-underscore	fleading-underscore	
	ftls-model=model	追加
	ftrapv	追加
	fbounds-check	追加

〔リスト12〕オプションなしで生成されたアセンブラソース(test194.s)

```
    .file     "test194.c"
.globl __cyg_profile_func_enter
.globl __cyg_profile_func_exit
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $main, (%esp)
    call __cyg_profile_func_enter
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test3
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $main, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    movl -4(%ebp), %ebx
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test1, (%esp)
    call __cyg_profile_func_enter
    movl $100, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test1, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test2, (%esp)
    call __cyg_profile_func_enter
    movl $200, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test2, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test2, .-test2
.globl test3
    .type     test3, @function
test3:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test3, (%esp)
    call __cyg_profile_func_enter
    movl $300, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test3, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test3, .-test3
    .ident    "GCC: (GNU) 3.3"
```

〔リスト13〕オプション付きで生成されたアセンブラソース(test194a.s)

```
    .file     "test194.c"
.globl __cyg_profile_func_enter
.globl __cyg_profile_func_exit
    .section  .rodata
.LC0:
    .string   "%d¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    andl $-16, %esp
    movl $0, %eax
    subl %eax, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $main, (%esp)
    call __cyg_profile_func_enter
    call test1
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test2
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    call test3
    movl %eax, 4(%esp)
    movl $.LC0, (%esp)
    call printf
    movl $0, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $main, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    movl -4(%ebp), %ebx
    leave
    ret
    .size     main, .-main
.globl test1
    .type     test1, @function
test1:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test1, (%esp)
    call __cyg_profile_func_enter
    movl $100, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test1, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test1, .-test1
.globl test2
    .type     test2, @function
test2:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test2, (%esp)
    call __cyg_profile_func_enter
    movl $200, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test2, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test2, .-test2
.globl test3
    .type     test3, @function
test3:
    pushl     %ebp
    movl %esp, %ebp
    pushl     %ebx
    subl $20, %esp
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test3, (%esp)
    call __cyg_profile_func_enter
    movl $300, %ebx
    movl 4(%ebp), %eax
    movl %eax, 4(%esp)
    movl $test3, (%esp)
    call __cyg_profile_func_exit
    movl %ebx, %eax
    addl $20, %esp
    popl %ebx
    popl %ebp
    ret
    .size     test3, .-test3
    .ident    "GCC: (GNU) 3.3"
```

「VxWORKS」を使ったRTOS技術の基礎と応用

第3回

ネットワークプログラミング――Zbuf&TELNET応用編

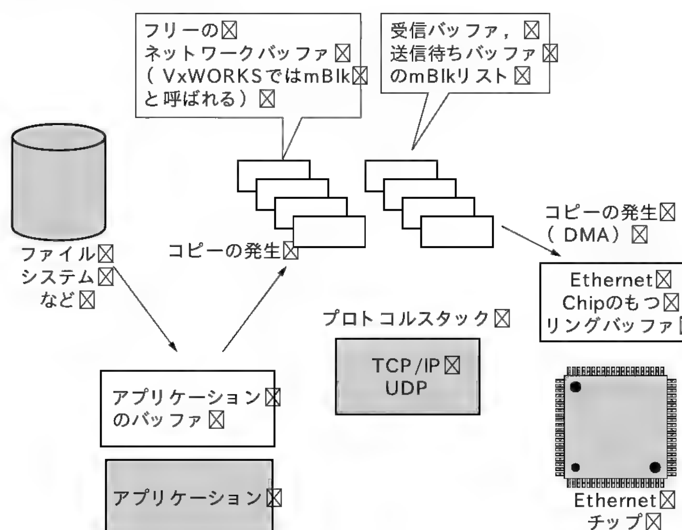
高山 剛

前回は，UNIX互換のソケットAPIを使ったWebサーバを紹介しました．ソケットAPIは互換性の高さから，組み込みシステムでも広く使われます．一方，UNIXと異なりRTOSでは，カーネルとアプリケーションの壁（プロテクション）がなく，カーネルサービスの呼び出しは単なるCのサブルーチンコールにすぎません．また実メモリ上に存在するメモリ資源は，そのままポインタによる相互参照が可能です．VxWORKSでは，この特性を生かしてZbuf（ZERO Copy TCP/IPと一般には呼ばれる）という，パフォーマンスを重視したAPIももっています．

もちろん通信相手がソケットAPIでも他のAPIでも，TCP/IPであれば通信が可能です．ただしZbufは，VxWORKS独自のAPIなため，そのままUNIXやWindowsへ移植できない互換性のなさがあります．また，コピーが減ったとしても，全体的なパフォーマンスを考えると大きな改善はないかもしれません．あくまで性能が要求される特殊な事情のあるときの選択肢の一つであることに注意が必要です．

今回は，前回のTHTTPDをZbufを用いて書き換え，実際のコードでそのコンセプトを解説し，後半はネットワークを介してコマンドラインのインターフェースを構築する際にtelnetdを応用して，独自のユーザーインターフェースをtelnetdに接続する例を紹介します．

〔図1〕標準のソケットAPI



Zbuf（ZERO COPY TCP）の応用

● ZERO COPY TCPとは？

図1のようにソケットインターフェースを用いると，少なくとも2度のデータのコピーが発生し，TCP/IPの通信においてボトルネックとなります．ソケットAPIとアプリケーションでは，ソケットAPIを生んだUNIXがカーネルとアプリケーションと別空間に存在するため，コピーという作業が必須だったのでしょう．しかしRTOSでは，カーネルとアプリケーションはスーパバイザで同一アドレス空間上で動くので，VxWORKSにはZbufというインプリメントのZERO COPY TCPをAPIとしてもっています．基本的アイデアとしては，アプリケーションで管理している複数のバッファをプロトコルスタックと共通の規則の範ちゅうで共有し，アプリケーションとネットワークプロトコルスタック間のデータコピーをなくすことを目的としています．このAPIを用いれば，アプリケーションとプロトコルスタック間でのデータコピーを不要にします．以下は，前号のthttpd.cに対する変更箇所です．

① #include "zbufLib.h"を挿入する

② LOCAL httpdSendFile ()関数を，次のように修正する

```
#ifdef ZBUF_VERSION
    char * pBuffer;
    ZBUF_ID zId;

    pBuffer = malloc (ZBUF_LEN);
    rtn = read ( fp, pBuffer , ZBUF_LEN );
#else   /* 変更前 */
    rtn = read ( fp, buf , sizeof ( buf ) );
#endif
```

③ read()でEOFとなったら，pBufferを解放する

```
#ifdef ZBUF_VERSION
    if  ( rtn <= 0 )
        {
```

```
                free ( pBuffer );
    #else
        if  ( rtn <= 0 )
            {
    #endif
```

④ LOCAL httpdSendFile ()関数で，write()に代わりzbufSockSend()を使って，次のように修正する

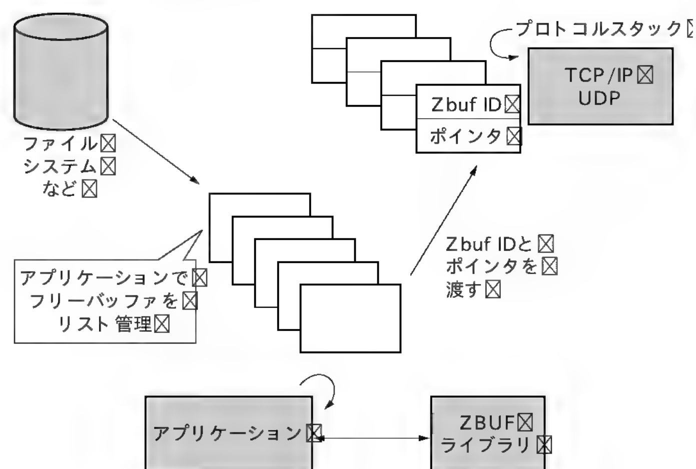
```
    #ifdef ZBUF_VERSION
            zId = zbufCreate ();
            zbufInsertBuf(zId, NULL , 0,
                        pBuffer , rtn, free , 0 );
            wvEvent(1,0,0);
            rtn2 = zbufSockSend ( sock , zId ,
                                         rtn, 0 );
    #else   /* 変更前 */
            rtn2 = write( sock , buf , rtn);
    #endif
```

図2およびこの変更箇所のCコードでその概要を説明します．ソケットAPIでは，ネットワークバッファはプロトコルスタック自身が管理し，アプリケーション自身が意識することはありません．しかしZbufによるZERO COPY TCPでは，アプリケーションで用意したバッファを送信用のネットワークバッファとなるようZbufのAPIを使って準備し，アプリケーションが直接データを埋めたら，そのポインタをプロトコルスタックに渡します．

②の修正は，アプリケーションで用意したバッファを送信用のネットワークバッファとして用意する修正です．とりあえず簡単に動作確認ができるだけの目的にmalloc()を使用しました．ただしこのままだと，メモリフラグメント(細分化)の原因(後述)にもなり，大量の送信を一度に行うとメモリを大量に消費してしまいます．malloc()/free()に相当するようなメモリの確保と解放のしくみをアプリケーションが用意して，用意したメモリが上限に達したときはバッファに空きができるまで待ち状態にさせるなどの工夫が必要になります．Zbufは，ソケットの汎用性，移植性を犠牲にして性能を追求するためのAPIなので，このアプリケーションバッファの設計もアプリケーションにゆだねられます．アプリケーションの性質やシステム要求にあった，それぞれの設計となると思います．必ず考慮しなくてはならないのは，システム中のメモリを消費し尽くさないように上限を決めることと，上限に達した場合は充分なネットワークバッファが有効になるまでタスクを待機状態にすることです．

④の修正は，図2のようにネットワークプロトコルスタックにZbufIDとバッファポインタを渡す必要があるので，Zbuf独自のAPI zbufInsertBuf()でまず，ZbufIDとバッファのポインタを関連付けます(リスト構造でZbufIDがリストの先頭ヘッダだと思えばよい)．このzbufInsertBuf()で重要な

〔図2〕Zbuf(ZERO COPY TCP)



のは6番目の引き数で，送信が完了した際，そのバッファを解放を行うコールバックルーチンを指定します．これはTCPの送信が，相手側にたしかに到着して着信確認を受け取るまで，プロトコルスタックがmBLK(プロトコルスタックでのネットワークバッファのこと．UNIXでのmbufに相当する)を破棄できず，アプリケーション側は，プロトコルスタックが破棄のタイミングでアプリケーションが指定したルーチンを呼び出してもらうわけです．次に，write()やsend()の代わりにzbufSockSend()を呼び出し，送信のリクエストを行います．

● CソケットAPIとZbufをビジュアルに比較する

VxWORKSの特徴の一つに，組み込みシステム向けに特化したツールがあることがあげられます．組み込みシステムでは，性能と信頼性が要求されます．一見性能は重要と思えないかもしれませんが，製品のコストを下げるには安価で低性能なCPU上で最適されたアプリケーションを走らせることで，製品の差別化が可能になります．ここでは，ソフトウェアアナライザと呼ばれる「WIND VIEW」で，ソケットAPIの場合とZbufの場合をビジュアルに比較してみました．

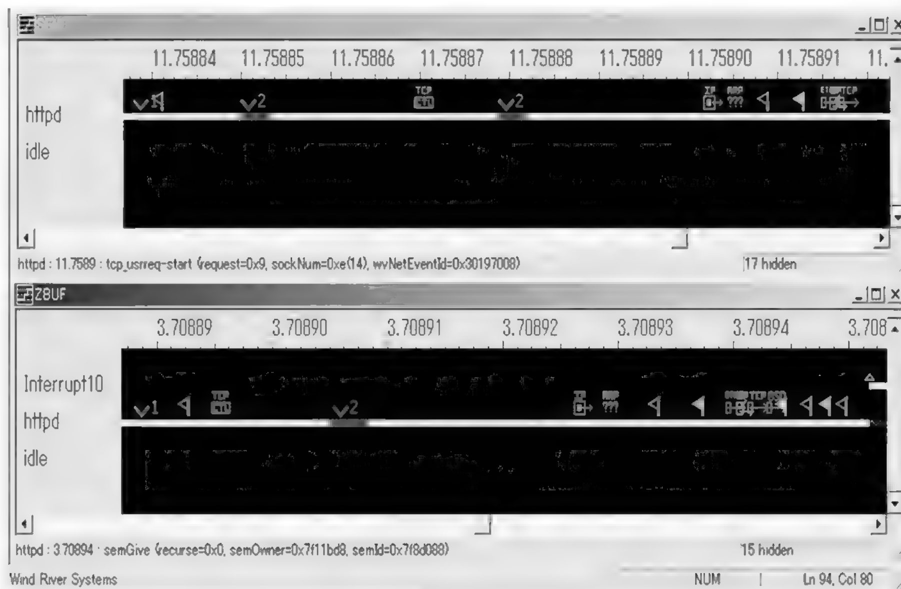
図3は，WIND VIEWで両者を比較したものです．記号V1はユーザーイベントと呼ばれるもので，プログラム中のwrite()の直前に，次のようにwvEvent()を挿入し，

```
    wvEvent(1,0,0)
    rtn2 = write( sock , buf , rtn);
```

アプリケーション上の特定の箇所でユーザー定義のイベントが発生するようにしました．ユーザーイベントを発生させると，後ほどWIND VIEWで検索分析する際に便利です．

“TCP TCL”，“IP”と文字とアイコンで示されているのはプロトコルスタックでのイベントが表されています(WIND VIEWは，デフォルトではネットワークのイベントは見られないので設定が必要．マニュアルにしたがってnetLibでのイベント収集をONにし，収集のレベルをVERBOSEにしてみてほしい．収

〔図3〕WIND VIEWでソケットAPIの場合とZbufの場合をビジュアルに比較する



集レベルには，潜在的問題を報告するだけのレベルからVERBOSEレベル：もっとも多くのイベントを報告するレベルまである）．たとえばTCP TCLの意味は，GUIの操作でNearest Eventを指定するとイベントの意味が解説されるようになっていて，「tcp_usrreq()でSOCK_STREAM(TCP)タイプのリクエストを受け付けるエンドポイントで……」となっています．

データコピーは，ネットワークイベントで報告されないので，拡張機能であるイベントポイント機能（e bcopy，イベント番号"でbcopy()の先頭にイベントポイントを張り，動的にイベントを検出してみました．V2と表示されているのは，"e bcopy, 2"と設定したためで，bcopy()にヒットしたことを示しています．

ご覧のとおり，ソケットAPIでは2度のデータコピー（TCP headerを生成するためのコピー，mBLKへのコピー，Ethernetチップへのコピーは実験に使ったボードがDMAを使っているので報告されていない）が発生しています．bcopy()付近に破線で示されているのは，bcopy()の実行時間ではなく，タスクロック中（プリエンプティブ禁止）を示していますが，おおよそbcopy()の処理時間と推測できます．このソケットAPIの例では，タスクロック期間中におおよそ2μsの時間がかかっていたことがわかります．

● UDPでのソケット以外のAPI

UDPでもZbufを使用可能です．さらにUDPの場合は，WIND RIVER Platform Network EquipmentにバンドルされているRouter StackのFastUdpLibを用いればさらに高速になります．Router Stackとは，一般のネットワーク端末のプロトコルスタック（ホストスタックと呼ぶ）と異なり，IPフォワーディングの効率に特化したルータ向けのプロトコルスタックです．

Router Stackについて詳しくは，White Paper（http://www.windriver.com/whitepapers/wp_pne.pdf）に記述されているのでご覧ください．

ただしFastUdpLibは，パフォーマンスと引き換えにソケットと比べると制限事項は多くなります．

独自シェルをtelnetdに接続する

最後に，ネットワークプログラミングの一つとして，PCやUNIXからTELNETを介してVxWORKSにログインしたとき，VxWORKSのターゲットシェルではなく，アプリケーション独自のシェルを接続する例を紹介します．

VxWORKSのデフォルトでは，TELNETでVxWORKSにログインを行うと，VxWORKS上で動作するTELNETのサーバであるtelnetdが，ラインデバッグ用のターゲットシェルに接続し標準入出力（デフォルトではシリアルCOM1）をリダイレクトとしてネットワークを経由したラインデバッグが可能です．このターゲットシェルはデバッグにおいてたいへん強力で，VxWORKSのOSやアプリケーションのあらゆるサブルーチンを呼び出すことが可能であり，メモリの書き換えなど，まったく制限がありません．UNIXでいえばルートの権限のみならず，ブートモニタまでネットワーク経由でアクセスできてしまうようなものです．このシェルはデバッグにはたいへん便利でかつ強力ですが，ユーザーアプリケーションのインターフェースとしては不適当です．

組み込みシステムのユーザーは，一般のユーザーを対象とせず，エンジニアである場合もありますが，かりにエンジニアであってもアプリケーションの知識を必要とさせても，OSの知識を前提とさせるわけにはいきません．エンドユーザーには，安全にあらかじめ設計された範囲内のオペレーションのみが許されなければなりません．

Column

VxWORKS ネットワークプログラミングに関するリンク集

◆ VxWORKS FAQ

http://www.xs4all.nl/~borkhuis/vxworks/vxworks.html

VxWORKSにまつわるFAQやフリーのコードも公開されています．その中に rshd.c が含まれています．

http://www.xs4all.nl/~borkhuis/vxworks/vxw_pt4.html#4.8-J

テストしたところ，すこし手直しするだけで問題なく動いているようです．コード中の，

```
if (qCmdPending) qCmdExec();
```

は，VxWORKSには相当する関数，変数がないため，アプリケーションの一部が欠落しているようですが，この行を削除しても動作上，問題ないようです．

rshの応用例としては，UNIXからVxWORKSへコマンド発行する際にcshのaliasと組み合わせると便利だと思います．

```
->alias vx121  'rsh 147.11.60.121'
->vx121 i
->vx121 pwd
```

かつて筆者がVxWORKS5.0のユーザーだったとき，いろいろ改造して，

```
->vx121 ld apl1.o
```

とすることで，いろいろなディレクトリからコマンド一つでダウンロードできるようにしていました．昔のコードが残っていないので公開できませんが，筆者の記憶では，socket経由のTCPで apl1.o をリダイレクトで送るだけでは，VxWORKSのローダでロードできませんでした．それは，VxWORKSのローダがSEEKを実行するからです［ソケットディスクリプタ(TCP)に対してはSEEKできない］．そこで ld，0，“絶対パス/filename”でコマンドを送ればファイルはリモートファイルシステムを経由するため，ダウンロードできるはずです．絶対パスを入れる点で，rshを改造していました．Target shellフリークの方には重宝するのではないでしょうか．

◆ WIND NET Tech Tips VxWORKS

https://www.windriver.com/windsurf/products/ide/tornado/t22/index.html

WIND RIVERのホームページには，顧客専用のオンラインサポートのWebサービスがありますが，この中にWIND NET Tech Tips VxWORKSで，ネットワークプログラミングについてヒントとなる，次のような情報がほかにも多数記述されています．

Muximum size for Socket Send/Receive Buffers，Socket as Non-Blocking, SOMAXCONN, SO_KEEPALIVE, MSS, SO_LINGER, Disable UDP checksum, ARP cashe size, IpFilterHook

◆ VxWORKSのフリーソフトウェア

ftp://ftp.atd.ucar.edu/pub/archive/vxworks/vx/index

VxWORKSで動作するX Window System，cursesなどが公開されています．VxWORKSでのプログラミング例の参照になるのではないでしょうか？ 一つ面白いものを紹介すると，loadmeter. というものがあります．CPUのIDLE時間を計測してCPUの負荷を計測するものですが，アルゴリズムは，最低優先順位のタスクを余分に走らせて無限ループ中でひたすら変数のインクリメンタルを行います．さらにウォッチドッグタイマにより一定間隔でその変数をモニタして差分のMAXを求め，そのMAX値をCPUの100％の性能と仮定して，現在のCPUの負荷を求めるというものです．VxWORKSには spy() というものがありますが，これはタイマ割り込みで，割り込みが発生しプリエンプティブしたタスクや割り込みコンテクスト，またはIDLE状態を特定してCPUの負荷を計測するものです．

Loadmeterのアプローチは原始的ですが，正確さではこのloadmeterが上です．ただし，このコードは68系でしか動作確認していないようで，少し修正が必要です．変数をインクリメントしている箇所が，68Kでは不可分な命令で置き換わりますが，RISCプロセッサでは，複数の命令で実現されるため，排他制御がなされません．なので，intLock()/intUnlock() で割り込みをロック/アンロックする必要があります．

組み込みアプリケーションの操作には，グラフィックを用いたGUIや，前回紹介したようなWebによるものがありますが，これはたしかにビギナ系のユーザーにはたいへんありがたいものですが，ヘビーユーザーのエンドユーザーになって来ると，コマンドラインのインターフェースが好まれる場合が多くあります．ルータなどではTELNETやシリアルでルータにログインしてコマンドラインインターフェースをもつものがほとんどのようです．

ターゲットシェル以外のユーザー独自のシェルを組み込む機能がVxWORKS5.5から，telnetdParserSet() により簡単に実現できるようになりました．この telnetdParserSet() を呼び出すか，マクロ TELNETD_PARSER_HOOK によって，より簡単に独自のシェルを組み込むことができます．

リスト1は，ダム端末，キャラクタ端末で使われるメニュー形式のごく簡単なユーザーインターフェースです．リスト2はリスト1をTELNET経由で使えるように修正したものと，パーサとtelnetdを関連付けるパーサコントロールルーチンから成っています．

リスト2での修正箇所は，

① リダイレクトできるように printf() を fdprintf() に置き換える

② telnetdExit ((UINT32)sessionId); で telnetd とのセッションの終了

③ ioctl (ioFd, FIOSETOPTIONS, oldoptions & ~(OPT_LINE | OPT_ECHO));

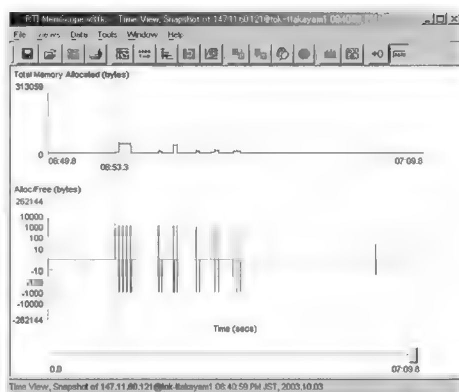
は，エコーバック，XON/OFFによるフロー制御などのシリア

〔リスト 1〕ごく簡単なユーザーインターフェース

```
void mainMenuOrg ()
  {
  char buf;

  FOREVER
     {
     printf (" ¥nXXXXX System Main Menu:¥n" );
     printf ("    [1] Configuration¥n" );
     printf ("    [2] Run ¥n" );
     printf ("    [3] Diagnostic system ¥n" );
     printf ("    Press 'Q' to quit.¥n" );
     printf (" --> " );
     fioRdString( STD_IN , &buf , 1 );
     switch ( buf )
       {
       case '1' :
       printf (" ¥nConfiguration:¥n" );
       fioRdString( STD_IN , &buf , 1 );
       break;
       case '2' :
        /* application code
         * Run system
         * your code
         */
        printf (" ¥nRuninng......:¥n" );
        fioRdString( STD_IN , &buf , 1 );
        break;
       case '3' :
    /* Diagnostic system
     */
        printf (" ¥nDiagnostic sub menu:¥n" );
        printf ("    [1] A sub-system¥n" );
        printf ("    [2] B sub-system¥n" );
        printf ("    [3] Sensor/Actuater¥n" );
        printf ("    Press 'Q' to quit.¥n" );
        printf (" --> " );
        fioRdString( STD_IN , &buf , 1 );
    /* your code */
        break;
       case 'Q' :
       case 'q' :
    return;
       }
     }
  }
```

〔図 4〕メモリリークの可能性のチェック



ル特有の設定を解除します．

リスト 2の myTelnet.c を VxWORKS で使用するためには，myTelnet.c を OS に組み込み，OS のコンフィグレーションを少し修正します．Tornado のプロジェクトから，target server のパラメータの一つである TELNETD_PARSER_HOOK（デフォルトは shellParserControl）をリスト 2の myParserControl（関数名）に書き換えます．同時に TELNETD_MAX_CLIENTS（デフォルトは 1）を 2に書き換えると telnetd のセッションを二つ，同時に起動することができます．さらに TELNETD_TASKFLAG（デフォルト FALSE）を TRUE にしておきます（TELNETD_TASKFLAG については後述）．これで，シリアル経由でターミナル端末で使っていたコマンドラインインターフェースを，従来どおりシリアルからでも，ネットワーク経由でどこからでも操作できます．

● メモリの細分化を防ぐ

このリスト 2のプログラムは，telnet で接続するたび，taskSpawn() で新しいタスクを生成するのではなく，セッションが終了するとタスクをサスペンド状態（強制的な待ち状態）にします．組み込みシステムでは，不用意なタスクの生成とデリートの繰り返しは避けなくてはなりません．さもないとメモリプール（ヒープ）のフラグメント（細分化）を起こし，長期間のシステム運用中に致命的な障害を起こす可能性を秘めています．たとえば，現在は IT エンジニアが TELNET で手動で設定を変えることを前提にしていても，自分の知らないところで，将来，他のコンピュータシステムから自動で TELNET を介して設定を頻繁に変えることを考えるかもしれません．

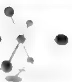
このリスト 2のプログラムは，TELNET で接続するたび，taskSpawn() で新しいタスクを生成するのではなく，セッションが終了するとタスクをサスペンド状態にします．再度 TELNET より接続をリクエストされた場合は，タスクをリスタートさせています．これで余分なタスクの生成とデリートを防いでいます．

前述した Tornado のプロジェクトから，target server のパラメータの一つ TELNETD_TASKFLAG（デフォルトは FALSE）を TRUE にしましたが，これはメインメニューとそれに付随する必要なタスク（入力/出力用それぞれのタスク）をあらかじめ立ち上げて生成とデリートの繰り返しを防いでいます．これでメモリフラグメンテーションに対しては万全です．

本当に万全であるかを確認するには，WIND VIEW でシステムレベルでシステムメモリ使用量をモニタして左右対称性（正常な場合左右対称性があるはず）があるかをチェックし，メモリリークの可能性のチェックが可能です．

図 4は，RTI 社の MemScope（CodeTEST/memory module）を使ってモニタしたものです．これらのツールを使えば，タスク単位での malloc/free の回数，消費量を時系列で比較，メモリリークを見つけた場合は malloc() からスタックトレースしアプリケーションのみならず，ミドルウェアや OS のメモリリークさえ発見が可能です．このようなツールがあると，メモリリーク探しが病みつきになるかもしれません．

たかやま・たけし　ウインドリバー（株）
takeshi.takayama@windriver.com



〔リスト 2〕リスト 1を TELNET経由で使えるように修正したものと，パーサと telnetdを関連付けるパーサコントロールルーチン

```
/*---------------------

  THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR
  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT,
  ARE DISCLAIMED. ARISING IN ANY WAY OUT OF THE USE OF
  THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

End of License.

---------------------*/

/* myTelnet.c - a telnet daemon for T2.2 */

/*
modification history
--------------------
01b,22oct03,tkt  rewritten
*/
/*
DESCRIPTION:
This example allows telnetd to connect specific user interface
for use with Tornado2.2

Instruction:

1  #define TELNETD_PARSER_HOOK myParserControl
   #define TELNETD_TASKFLAG TRUE
   and build this module with your kernel

2. Re-build vxWorks kernel.

3. Telnet to your target.

example
unix-> telnet target

*/


#include "vxWorks.h"
#include "taskLib.h"
#include "telnetLib.h"
#include "ioLib.h"

void mainMenuTelnet (int ioFd, TELNETD_SESSION_DATA * sessionId)
     {
     int oldoptions = ioctl (ioFd, FIOGETOPTIONS, 0);

     ioctl (ioFd, FIOSETOPTIONS, oldoptions & ~(OPT_LINE |
OPT_ECHO));
     mainMenu (ioFd);
     telnetdExit ((UINT32)sessionId);
     taskSuspend(0);
     }

void mainMenu (
    int ioFd
    )
    {
    char buf;


    for (;;)
       {
       fdprintf (ioFd, "¥nXXXXX System Main Menu:¥n");
       fdprintf (ioFd, "   [1] Configuration¥n");
       fdprintf (ioFd, "   [2] Run ¥n");
       fdprintf (ioFd, "   [3] Diagnostic system ¥n");
       fdprintf (ioFd, "   Press 'Q' to quit.¥n");
       fdprintf (ioFd, " --> ");
       fioRdString( ioFd , &buf , 1 );
       switch ( buf )
       {
       case '1':
        fdprintf (ioFd, "¥nConfiguration:¥n");
        fioRdString( ioFd , &buf , 1 );
        break;
       case '2':
        /* application code
         * your code
         */
        fdprintf (ioFd, "¥nRuninng......:¥n");
        fioRdString( ioFd , &buf , 1 );
        break;
       case '3':
    /* Diagnostic system
     */
        fdprintf (ioFd, "¥nDiagnostic sub menu:¥n");
        fdprintf (ioFd, "   [1] A sub-system¥n");
        fdprintf (ioFd, "   [2] B sub-system¥n");
        fdprintf (ioFd, "   [3] Sensor/Actuater¥n");
        fdprintf (ioFd, "   Press 'Q' to quit.¥n");
        fdprintf (ioFd, " --> ");
        fioRdString( ioFd , &buf , 1 );
    /* your code */
        break;
       case 'Q':
       case 'q':
        return;
       }
     }
   }


STATUS extern myParserControl
  (
  int telnetdEvent,                  /* telnet session    */
  TELNETD_SESSION_DATA * sessionId, /* session identifier */
  int ioFd
  )
  {

   if (telnetdEvent == REMOTE_INIT)
     {
     return (OK);
     }
   else if (telnetdEvent == REMOTE_START)
     {

     if ( sessionId->shellTaskId == NULL )
        sessionId->shellTaskId = taskSpawn ("myTelnetd", 100, 0,
           20000,(FUNCPTR)mainMenuTelnet, ioFd, (int)sessionId,
                                                 0,0,0,0,0,0,0,0);
     else
        taskRestart ( sessionId->shellTaskId );

     return (OK);
     }
   else if (telnetdEvent == REMOTE_STOP)
     {
     return (OK);
     }
   }
```

組み込みLinuxをとりまく世界

第4回（最終回）　組み込みLinuxのミドルウェアとその評価環境　渡辺 武夫

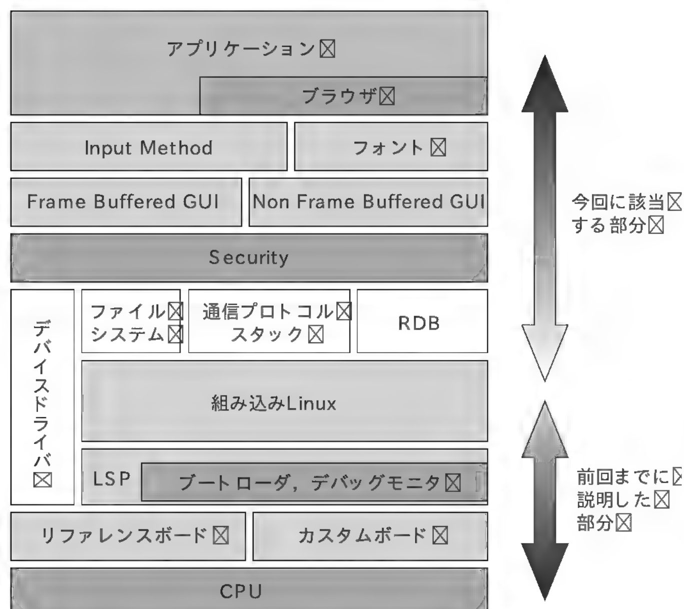
はじめに

前回までに，組み込みLinuxを使用していくうえで必要となってくるプラットホームの構築，評価，Webサーバ構築などを，「Embedded Linux Reference Kit（ELRK）」をベースに解説してきた．最終回にあたる今回は，実際の組み込み機器開発時に使用されるミドルウェアの現状と，その内容について解説する．

ミドルウェアについて

現在開発されている組み込み機器は，各マーケットによって複雑化し，使用されるミドルウェアの種類，量ともに増加傾向にあるといえる．CPUからアプリケーションにいたるまでの構造をまとめると，図1のようになる．図1で「今回に該当する部分」と示されているところが，ミドルウェアである．

〔図1〕ミドルウェアの位置づけ



この中でとくに通信プロトコルについては，IPv6，VoIP，ルーティング，802.11b，802.11a，802.11gなど多くの種類が使用され，各プロトコルの評価，実装について，多くの開発工数がさかれている．また組み込みLinuxでは，オープンソースベースのスタックとサードパーティなどから販売されているスタックを比較検討しているケースも多く，より多くの時間を費やすこととなる．表示系については，フレームバッファ対応のものとフレームバッファを使わないものとに大別される．フレームバッファとしては，GTK，Qt Embeddedなどが使用される場合が多いが，日本語表示やインプットメソッドとの連携についての対応もさまざまである．フォントでは，X Free Type Library，Free Type Libraryへの対応状況については各フォントベンダの対応に依存しており，開発側が検討し，各ベンダにそのつど，オーダーするスタイルとなっている．

ミドルウェア評価の現状

ミドルウェアの評価は，プラットホームへのOSレイヤのポーティングから始まる．ここでは，プラットホームの周辺機器を動作させるためのブートローダ開発，ドライバ開発などの作業が発生し，プラットホームの安定稼動後，各種ミドルウェアの評価を行っていくこととなる．現状では，評価環境の共通化が行われていないため，各プラットホームに対してのミドルウェアのポーティングに関しては，ソフトウェアベンダに依頼することとなる．その際，SDKの購入，ポーティング費用などが発生する．同一カテゴリのソフトウェアを比較評価するような場合には，それぞれに対してSDKの購入とポーティング依頼を行う必要があり，比較対照の結果，使用しなかったものに関しては，不良資産化するおそれがある．製品出荷までのおおよその流れを図2に示す．このような期間を短縮し，開発者の負荷を軽減するには，各ミドルウェアベンダが協調した評価プラットホームが必要となってくると考えられる．

ソフトウェアベンダによるミドルウェアの例

ここからは，現在使用されている通信プロトコル，日本語環

境のミドルウェアの例をいくつか解説する．

● SSH Communication Security（SSH社）のSSH QuickSec Toolkitについて

近年，ネットワークがどこにでもある状況となり，同時にさまざまな情報がネットワークに流れてくるようになった．このような状況で真っ先に考えられるのが，“機密保護”をキーワードとしたセキュリティということになる．ここでは，このセキュリティをキーワードとした製品である“SSH QuickSec Toolkit”について説明していく．SSH QuickSec Toolkitのすべてを記すと膨大な量になってしまうので，とりあえずいくつかの代表となるキーワードを記してみる．

- IPsec　：通信データ（パケット）のカプセル化などを実現
- IKE　：ピア認証やIPSecトンネル確立を実装するために使用
- Firewall：文字どおり，通信のブロックを行う
- NAT　：通信相手先（アドレス）の変換を実現

これ以外にも，ネットワークに関わる通信についてのさまざまな機能が盛り込まれている．詳しくは同社のWebページで紹介されているので，そこを参照してほしい（http://www.ipsec.co.jp/）．

▶ フリーソフトウェアとの違い

同社の製品の多くは，オープンソフトウェアやフリーソフトウェアで，無償で手に入れることが可能である．つまり，これらの機能を使用したければ何もお金をかけなくとも入手可能なので，一見，製品として存在していることの意味がないような気もしてくるのだが，これが大きな問題で，実際のところ，個人でオープンソフトウェアを入手した場合，①構築ができない（動作しない），②（セキュリティの）責任をもてない，③アップデート（IEEE規格など）との連携ができない，といった問題を抱えてしまうことになる．

したがって，ただ個人的に面白半分や，それ自体の内部研究であればオープンソフトウェアやフリーソフトウェアを入手して試すのも手だが，情報が分散しており，非常に膨大な書物を読み漁ってやっと実現できるのが現在の状況だったりする．また，その間に最近流行のセキュリティホールの発覚や，規格の変更が発生し，せっかく組み込んだソフトをアップデートしなければならず，組み込み作業ですべての時間を費やしてしまう場合もあるのではないだろうか？　実際の製品を意識したものに使うのに二の足を踏んでしまうことはないだろうか？　このように考えていくと，製品として保証されたものを使用することが選択肢の一つとして浮かんでくる．

▶ Linux環境での動作

具体的にLinux環境で実際にセキュリティ機構を使用する場合，どのような順番で，どのような機能を追加したらよいかが最初の疑問となるはずである．SSH QuickSec Toolkit 注1 は，図3に示すBuilding Blockを基準に，セキュリティスタック部分で必要な各種ライブラリなどを，ひとまとめで提供している．

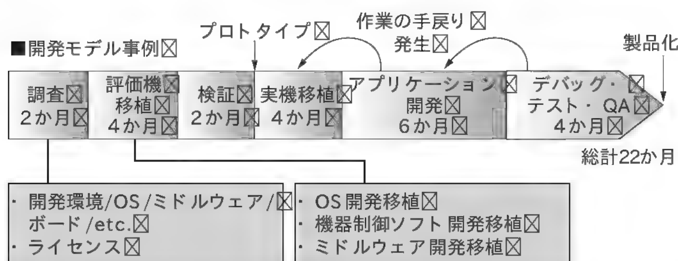
▶ SSH QuickSec Toolkitを用いた場合の特徴

SSH QuickSec Toolkitには，IPsecの実装に必要なものすべてが入っている．したがって，OS動作環境と一般的なネットワーク動作環境（TCP/IPなど）が実現されている環境であれば，とくに本ツールキット以外に用意するべきものはない．また，実際にパケットカプセリングを行う場合，ネットワーク機構（プロトコルスタック）そのものに細工を施す（カプセリングファンクションの追加）必要があるため，その作業が複雑になりがちである．しかし，図4に記すように，既存のネットワーク機構にくさび型にインターセプタを挿入し，それが実際のカプセル化エンジンと通信を行い，カプセル処理を実現させるようにしている．この構造を用いることにより，このエンジンのパラメータを外部から与えてカプセル条件定義を行ったり，エンジン部分をハードウェアにまかせることにより，通信速度維持（一般的に，IPsec機構を入れると通信パケットのカプセル作業があるため通信速度が低下する）を実現させることなどが可能となっている．

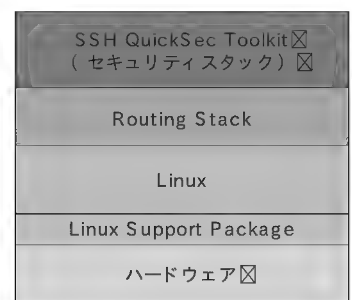
● IP Infusion社のZebOSとは？

ネットワーク環境が増え，セキュリティの必要性があることを先に述べたが，それにあわせて「ルーティング」という言葉も持ち上がってきた．実際のところ，ネットワーク環境ができあがると各端末が容易に接続（通信）できるようになるが，実際に

〔図2〕組み込みシステムの製品出荷までの流れ



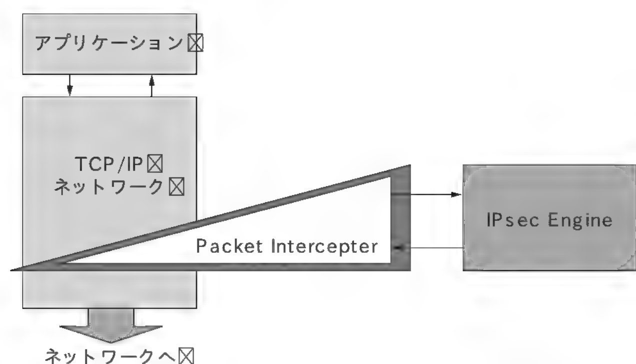
〔図3〕SSH QuickSec Toolkitの位置づけ



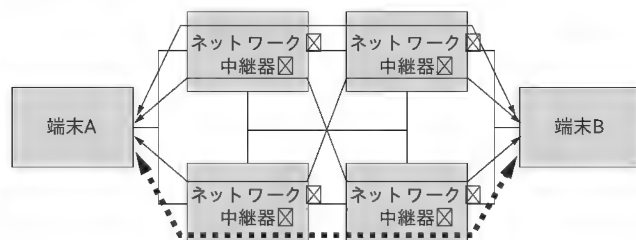
注1：SSH QuickSec Toolkitの開発元であるSSH Communication Security Corp.は，OEMビジネスを米国SafeNet社へ売却することとなり，今後QuickSecに関する開発，マーケティング，営業，サポート業務はSafeNet社へ引き継がれる．

は一本線でつながっているわけではなく，さまざまな経路（ルート）で接続できるようになっている．ただし，これはあくまで理論的な話であり，実際の通信の世界では最初に通信が実現されたルートを常に使用していることがほとんどである（図5）．通常はこのような条件で問題ないが，動作中のネットワーク中継器が何かしらの問題で故障した場合，通信が切断されてしまう．とくに，最近流行のIP電話などを用いた場合，中継器の故障やメンテナンスなどの理由で切断されると，使用者の意志とは無関係に電話が切れてしまい，実際には使えないものとなってしまうことになる．このような問題を解決させるには接続経路を監視しつつ，問題発生時（経路切断）には他方の経路に自動的に切り替える（スイッチング）システムを組まなければならない（図6）．IPInfusion社（http://www.ipinfusion.com/jp）のZebOS[注2]はこのような，ルート（ルーティング），スイッチング機能を実現させるためのソフトウェアである．

［図4］インターセプタの挿入

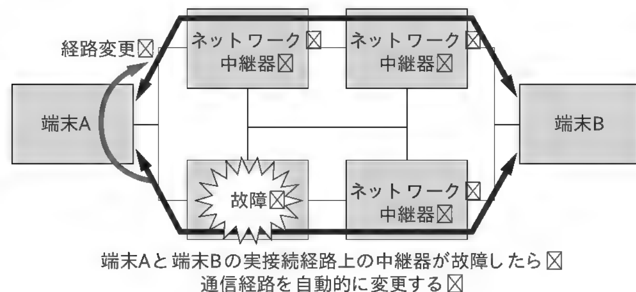


［図5］通常の通信



端末Aと端末Bをつなげる道筋は一つではない．
しかし，最初につながった道筋が点線の場合，その経路で通信が行われる

［図6］経路変更



端末Aと端末Bの実接続経路上の中継器が故障したら
通信経路を自動的に変更する

▶ZebOSの特徴

ZebOSの特徴としては，“ダイナミックルーティング”をキーワードとした，前述した通信経路の自動切り替え機構をはじめ，非常に多くの特定ネットワーク機構との親和性があげられる．たとえば，先に述べたSSH社のセキュリティ機構と組み合わせ，堅固なVPN環境を作り出せる．また，ZebOS搭載環境同士の自動検出なども可能で，結果として新たに構築された通信経路を自動識別させることもできる．ZebOS自体はネットワーク上すべての機器に必ずしも入れる必要はなく，通信状態を保障させたい箇所に実装することで実現可能である（図7）．したがって，一例としては，すでにZebOS＋VPNで環境構築が完了した，本社/支店間を結ぶネットワーク環境に，支店追加にともなうインフラ整備のためのネットワーク停止を理論上は無視することも可能である．

● 日本語環境の例（Justsystem ATOK）

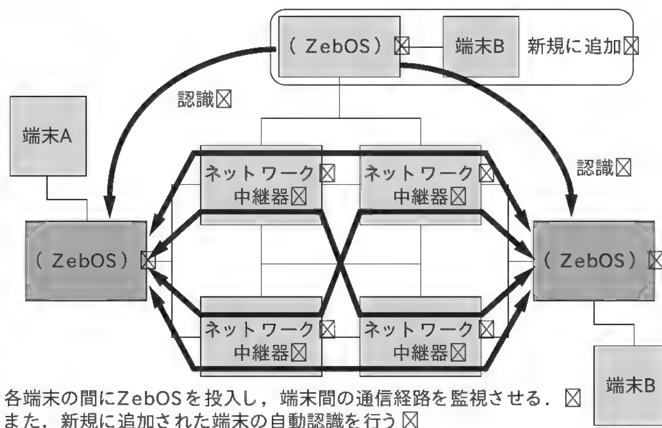
Linuxでは X Input Method（以下XIM）がX11R5から導入されて以降，現在でも各ベンダから提供される商用Linuxには標準的なIMとして実装されている．しかし，XIMはその名前のとおり，X Window Systemの内部に完全に組み込まれた日本語入力システムとなっており，そのためこの入力システムを利用するためには，どうしてもXを切り離すことができなくなっている．

最近は，Qtopia，GTK+-2.0（Linux Framebuffer），PowerPartsのように，Xに依存することのないグラフィックライブラリが存在し，実際組み込み機器に実装される例も多数出てきた．しかしながらこれらのグラフィックライブラリはXに依存しないため，XIMを利用できないのが現状であり，日本市場向けの製品を生産する際のいちばんの問題点となる．

今回の実装では，この問題を解決するために，グラフィック

注2：「ZebOS」とは商品名であり，Linuxなどの上で動くソフトウェアパッケージである．

［図7］ZebOS



各端末の間にZebOSを投入し，端末間の通信経路を監視させる．
また，新規に追加された端末の自動認識を行う

〔表1〕ATOKエンジン

ATOKエンジン実装機能	サイズ
連文節変換	800Kバイト～ (辞書データとATOKエンジンの総計)
口語体対応	
単語登録	
学習	
確定アンドゥ	
推測変換	

ライブラリにまったく依存しないIMをコンポーネントの一つとして提供している．このIMは，XIMのようにXに依存することがないため，前述したXに依存しないグラフィックライブラリ上へ実装も可能である．提供するInput Method(以下，IM)の中核となる変換エンジンには，実装機能・変換エンジンのサイズ・変換効率などの点(表1参照)から，(株)ジャストシステムのATOKを内部に組み込み，IM自体のサイズ縮小化と変換効率の向上を図った．

▶ フレームワーク

IMは，クライアント-サーバモデルを採用している．サーバサイドとなるIMはループバックデバイス(loデバイス)を利用してシステム上のデーモンプロセスとして起動した後，クライアントサイドとなるアプリケーションからのIM利用要求を待つ．アプリケーションはサーバと接続するためのライブラリを呼び出すことにより，サーバとの回線接続・データ通信・回線切断を行うことが可能となっている．また極端な例として，サーバサイドのループバックデバイスをEthernetデバイス(eth0デバイスなど)に変更することにより，クライアントサイドとなるアプリケーションは遠隔地に存在するIMと接続・通信することも可能となっている．

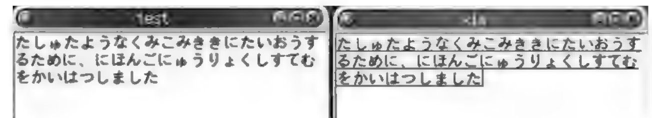
▶ 拡張性

現在，多種多様な組み込み機器が存在しているが，今後の組み込み機器はさらに多種多様な組み込み機器が開発・生産されると考えられる．そのため，提供するIMは，IM自体は非常に単純な構成(クライアントサイドであるアプリケーションからの接続待ち・データ通信・日本語変換システム・接続切断)となっているが，IMに対してPluginを追加することにより，IM自体の機能拡張を行える．たとえば現在一般的な入力は，ハードウェアキーボード・ソフトウェアキーボード，もしくは手書き入力となっているが，音声認識などといった機能をPluginとしてユーザーが作成することにより，IMは音声認識が可能なIMとして動作する．

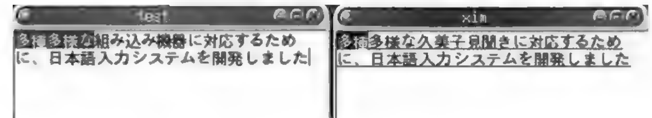
▶ 柔軟性

日本語対応のアプリケーションを作成するうえで，盲点となると思われるのが文字コード問題である．ベンダが提供するIMは，UTF-8・EUC・シフトJISなどの文字コードに対応する必要がある．これらの文字コードに対応することにより，現状では動的に変更できないが，IMを実装するシステムにあわせ

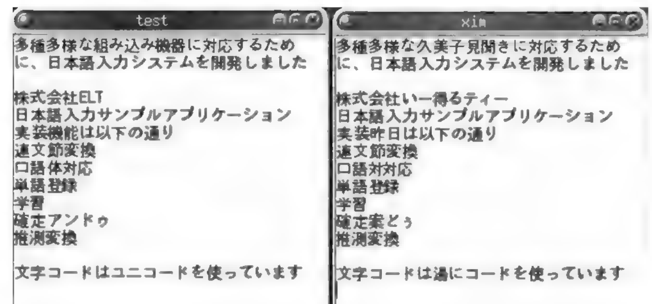
〔図8〕初期入力



〔図9〕変換キーを一度だけ入力した結果



〔図10〕複数の単語を同じ条件で変換した際の変換結果



た文字コードを選択することが可能となる．

▶ サンプル

図8～図10は，今回の実装で提供するIMのサンプル画像である．OSはRed Hat Linux7.2，グラフィックライブラリはGTK+-2.0(X Window System)を使用している．サンプル内ではATOKを変換エンジンとして実装したIM(左側)と，Red Hat Linux7.2標準のIM(右側)との変換効率の違いを示している．図8のように"たしゅたようなくみこみききにたいおうするために，にほんごにゅうりょくしすてむをかいはつしました"と入力後，変換キー(一般的にスペースキー)を一度だけ入力した際の結果が図9である．また，図10は，複数の単語を同じ条件(入力後変換キーを一度だけ入力)で変換した際の変換結果である．通常のLinuxでの変換結果は，ATOKの変換と比較した場合，誤変換が多くあることがわかると思う．

おわりに

4回にわたり，「組み込みLinuxをとりまく世界」と題して，解説を行ってきた．今後，組み込みLinuxの使用率は増加し，開発環境，評価環境，ミドルウェアに求められる要求も多様化していく．これらのニーズに対応するためには，ユーザーが正しく比較評価できるための統一化された評価プラットホームが必要であり，組み込みLinuxをとりまく各ソフトウェアベンダの製品供給方法の変化，ソフトウェアベンダ間の協力体制の変化が求められるであろう．

わたなべ・たけお　(株)イーエルティ



第24回

C/C++に似た言語仕様をもつスクリプト言語 ―― Pike

水野 貴明

今回紹介するのは，Pikeというスクリプト言語である．これは前回紹介したD言語と同様，CやC++によく似た書式をもつ言語である．Pikeはゲームを記述するための言語として誕生したが，その後，汎用的な言語として進化を続けている．

 インストールと実行

Pikeは，スウェーデンのリンシェーピン大学の情報工学科でメンテナンスされており，配布もそのWebサイト上で行われている．ソースコードのほかに，Linux（x86/PPC），Solaris（SunOS），およびWindows版のバイナリもダウンロードが可能だ．今回は，WindowsとLinuxにそれぞれのバイナリ版をインストールしてみた．

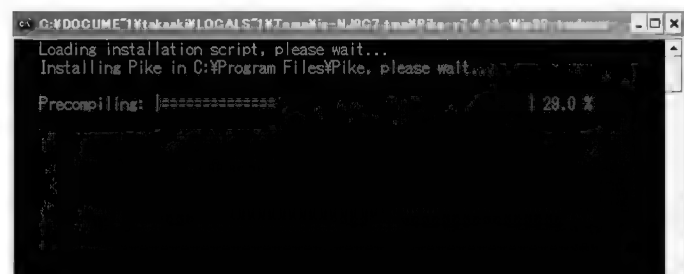
● Windows版のインストールと実行

まずはWindows版である．Windows版のバイナリはベータ版で，動作が若干不安定なだけでなく，ソースコードや，UNIX系OS向けのバイナリと比較すると，バージョンのアップデートが遅れる場合があるようだが，GUI版のインストーラとしてまとめられているので，インストール作業は非常に簡単である．

インストール作業では，ファイルをコピーし，「.pike」という拡張子をPikeの実行ファイルに関連付けた後，コマンドプロンプトが立ち上がり，ライブラリのプリコンパイルが行われる（図1）．

パスは自動的に追加されないので，インストール終了後，環境変数`path`にPikeの実行ファイルのパス（`C:¥Program Files¥Pike¥bin`など）を追加しておくとよいだろう．

〔図1〕インストール中．プリコンパイル処理が行われる



DATA

名称：Pike
作者：Pike development team
Webサイト：`http://pike.ida.liu.se/`
現在のバージョン：7.4.27（2003年9月18日現在）
ダウンロードサイズ：6.5Mバイト（ソースコード）
実行：インタプリタ
OS：Windows/UNIX

● Linux版のインストールと実行

続いて，x86のLinux版のバイナリをインストールしてみた．Linux版のバイナリは，巨大なシェルスクリプトとなっており，ファイルをダウンロードし，実行権限をつけた後，管理者権限でこのスクリプトファイルを実行すればよい．実行すると，インストールパスとPikeの実行ファイル名を尋ねられた後（標準は`/use/local`と`/usr/local/bin/pike`），ファイルのコピーが行われ，Windows版と同様，ライブラリのプリコンパイルが行われる．

プログラムの実行は，PerlやRubyなど，ほかのスクリプト言語と同様，Pikeの実行ファイルにスクリプトファイルを引き数として渡してやればよい．

```
> pike test.pike
```

またUNIX系OSでは，スクリプトファイルに実行権限をつけてそのまま実行したり，Windows版の場合はスクリプトファイルを直接起動することでも実行できる．

 Pikeの基本的な言語仕様について

簡単なPikeのプログラムをリスト1に示す．これは指定したWebサーバにアクセスし，指定されたURLのデータをダウンロードして表示するプログラムだ．取得するURLは，パラメータとしてプログラム実行時に指定可能なほか，指定されていなかった場合は，プログラム中で入力が求められるようにもなっている．

Pikeのプログラムは，C/C++と非常によく似たスタイルを



Column　Pikeの歴史

Pikeはもともとゲーム開発用の言語だったLPCという言語を基にして作られたという，ユニークな経歴をもつ言語である．その誕生と歴史を，Pikeのサイトに置かれたドキュメント注Aからひもといてみることにしよう．

LPC(Lars Pensjo C)は，1989年にスウェーデンのエーテボリにあるチャルマース工科大学にいたLars Pensjo氏によって生み出された，MUD(Multiple-User Dungeon)と呼ばれる，テキストベースの多人数プレイが可能なロールプレイングゲームを開発するためのプログラミング言語である．

それから約1年後，LPCで作られたゲームのユーザーの一人であったFredrik Hubinette氏は，LPCが非常にわかりやすい言語であることに気づき，それを独自に拡張して「LPC4」という名前を付けた．LPC4はやはりゲーム開発用の言語だったが，さまざまな拡張のおかげで，それ以外の目的にも耐え得る言語へと成長していく．しかし，LPC4はLPCのコードを利用していたため，商用利用してはいけないというLPCのライセンスにしばられていた．そこでFredrik Hubinette氏は「μLPC」という，LPCとほぼ同じ言語仕様をもち，ライセンスとしてGPLを採用した新しい言語の開発を開始することにした．

そのころ，InformationsVavarna AB社という企業が，LPC4を使って非商用のWebサーバを開発していた，やがてμLPCが利用に耐え得るものになったのを機に，InformationsVavarna AB社はその開発環境をμLPCに乗り換えた．そして1996年，Fredrik Hubinette氏はInformationsVavarna AB社(現在のRoxen Internet Software社)のために，μLPCを開発するまでになった．そしてμLPCは，より親しみやすいPikeという名前に変えられたのである．ちなみにPikeとは「槍」という意味もあるが，ここではカワカマスという魚のことを指す．Pikeのロゴも魚になっている．

その後，2002年に同じくスウェーデンのリンシェーピン大学の情報工学科がメンテナンスを引き継ぎ，Pike development teamが組織されて(Fredrik Hubinette氏もメンバとして参加している)，現在に至っている．

注A：http://pike.ida.liu.se/about/pike/history.xml

している．まず，プログラムのエントリポイントは`main`関数である．それぞれの命令文はセミコロン(;)で区切られる．`if`文や`do`文などの制御構造もほぼ同じである．ブロックを{ }で囲む点や，関数の定義の方法なども，C/C++とほとんど同じといえるだろう．C/C++を使った経験があれば，プログラムの記述は，ほとんど問題なく行うことができるはずだ．

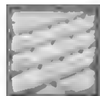
リスト1では，メッセージに日本語を利用している．日本語に関しては，今回検証を行ったWindows/Linuxでは，Windowsの場合はシフトJIS，Linuxの場合はEUC-JPを利用した場合は，日本語表示も正しく行えた．なおPikeは，スクリプトをUnicode系の文字コード(UTF-8/16)として保存することも可能である．日本語を使うのであれば，Unicodeを使ったほうが問題が少なそうだが，日本語を含むソースコードをUnicodeで保存してしまうと，実行時にエラーが出てしまう．これは，Pikeが内部的にはISO-8859-1を利用しているためだ．Unicodeで記述されたプログラムは，プリプロセッサによってコード変換が行われるが，その際に日本語の文字が入っていると，マッピングを行うことができずにエラーとなってしまう．

〔リスト1〕Pikeのサンプルプログラム

```
#!/usr/local/bin/pike
import Protocols.HTTP;

int main(int argc, array(string) argv)
{
  write("--- シンプルブラウザ ---¥n");
  string url;
  if(argc == 1) {
    do {
      write("表示するURLを入力してください:¥n");
      url = Stdio.stdin->gets();
    } while(sizeof(url) == 0);
  } else if(argc == 2) {
    url = argv[1];
  } else {
    write("引き数が多すぎます.¥n");
    return 1;
  }
  handle_url(url);
  return 0;
}

void handle_url(string this_url)
{
  write("データ取得中... '" + this_url + "'...");
  Query web_page = get_url(this_url);
  if(web_page == 0) {
    write(" 失敗しました.¥n");
    return;
  }
  write("終了.¥n");
  write("データ取得成功 '" + this_url + "':¥n¥n");
  string page_contents = web_page->data();
  write(page_contents + "¥n");
}
```



Pikeにおける変数の扱い

それでは，Pikeの言語仕様について見ていくことにしよう．まずは，変数についてである．Pikeでは，PerlやRubyなどのスクリプト言語と異なり，変数はあらかじめ定義してからでないと使えない．利用できる変数型を表1に示す．定義の方法は，基本的にはC/C++と同様，変数型を先に書き，その後に変数名を書く．

```
int dat;
```

ここまでは，C/C++とまったく同じである．しかしPikeでは，次のように指定することで，一つの変数に複数の変数型を割り当てることもできる．

〔表1〕Pikeで利用可能な変数型

int	整数
float	浮動小数
string	文字列
array	配列
mapping	連想配列
multiset	連想配列のキーのみの集合
program	Pikeのクラスを格納する
object	オブジェクト参照
function	関数への参照
mixed	すべてのデータを格納可能

```
int|string dat;
```

複数の型を割り当てることで，その変数にはそのどちらのデータも格納できるようになる．また，mixed型を指定すれば，どんなデータでも入れられる変数を作ることも可能だ．変数内で利用する変数型を厳密に指定するようにしながらも，複数のデータ型を割り当てられることで，柔軟に拡張できるようにしている点はなかなか面白い．

なおPikeの整数型は，GMP(Gnu Multiple Precision)ライブラリ注1を利用することで，非常に大きな値を扱うことも可能になっている．次の処理を実行すれば，1234567890の4乗である「2323057227982592441500937982514410000」という値を得ることができる．

```
int i = 1234567890;
int j = pow( i,4 );
write( "%d¥n",j );
```

さて，Pikeにおいては，すべての変数の初期値は0(ゼロ)である．しかもこれは，数値型変数だけではなく，文字列型の場合も同じである．しかも，初期化されていない変数はすべて整数型と認識されてしまうようなので，変数の初期化を行わないと，次のように関数にパラメータとして渡す際に，文字列型を渡しているにも関わらず，「expected string, got int.(文字列型の代わりに整数型が渡された)」というエラーが発生してしまう．

```
string s;
write("%s¥n",s);
```

そこで，定義の際にきちんと初期化しておく必要がある．

```
string s="";
```

続いて，複数のデータを格納できるデータ型(コンテナ型)である．Pikeでは，コンテナ型として用意されているデータ型は「array」,「mapping」,「multiset」の三つである．

まずはarrayだが，これは一般的な配列である．

```
array a = ({ 17, "hello", 3.6 });
write( a[1] );
```

上記の例では，配列の要素は複数の変数型が混在している．

注1：http://www.swox.com/gmp/

これは，上記のように単にarray型として変数を定義した場合，その要素のデータ型が「mixed」に設定されるからである．要素の変数型を制限するためには，定義の際にそれを指定する．

```
array(int) a = ({ 1, 2, 3 });
```

Pikeにおける配列は，動的に要素数が拡張されることはないので，範囲外の要素にアクセスしようとした場合，エラーとなってしまう．ただし，次のように初期化をやり直せば，範囲を設定しなおすことはできる．

```
array a = ({ 17, "hello", 3.6 });
a = ({ 1, 2, 3, 5, 6, 7, 8, 9, 0 });
```

続くmappingは，文字列をキーとして利用することが可能な連想配列である．mapping型はarray型と異なり，動的に要素を追加することができる．

```
mapping m = ([ "one":1, "two":2 ]);
m["three"] = 3;
write( "%d¥n",m["three"] );
```

Pikeでは，mapping型のデータ同士の足し算や引き算などを行うこともできる．その場合，たとえば次のようになる．

```
mapping m1 = ([ "dog":1, "cat":2 ]);
mapping m2 = ([ "cat":5, "lion":3 ]);
mapping m3 = m1 + m2;
        // ↑([ "dog":1, "cat":5, "lion":3 ])

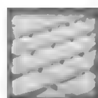
mapping m4 = m1 - m2;  // ← ([ "dog":1 ])
mapping m4 = m1 & m2;  // ← ([ "cat":5 ])
```

これを利用すると，特定のキーの情報だけを消したり，必要なキーだけを残すといったことも簡単にできる．

最後にmultisetだが，これはmapping型のキー情報だけを保存するような変数型である．この変数型は，袋の中にデータを放り込むように，さまざまなデータをまとめて保持しておきたい場合に利用する．multiset型では，同じキーが複数回存在することも許される．

```
multiset ms = (< "", 1, 3.0, 1 >);
```

multisetをどういった用途に利用すべきかは，なかなかわかりにくいが，arrayが動的な要素追加ができないため，次々とデータをストックしておき，最後にまとめて処理を行うようなプログラムの場合には，配列の代わりに利用することができるだろう．



オブジェクト指向プログラミングへの対応

Pikeはオブジェクト指向プログラミングに対応しており，クラスを定義して利用することができる．クラスの定義は，リスト2のようにして行う．createという関数はコンストラクタである．リスト2では，コンストラクタで文字列を受け取り，nameという変数にセットしている．ちなみにデストラクタはdestroyという関数として定義する．

〔リスト2〕クラスを定義する

```
class dog
{
  string name;
  void create(string n) {
    name = n;
  }
  void bark() {
    write("Bow wow!¥n");
  }
}
```

クラスを利用する場合は，次のようにしてインスタンスを作成する．

```
dog my_dog = dog("pichi");
```

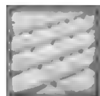
メンバにアクセスするには「->」を利用する．

```
my_dog->bark();
```

クラスの継承を行う場合は，inheritを利用する(リスト3)．Pikeでは多重継承も可能である．その場合は，inheritを連続して記述する．

さらに，メンバの定義の前にリスト4のように「private」「final」のようなキーワードをつけることで，そのメンバの属性を設定することも可能である．privateを指定すれば，外部からアクセスができなくなり，finalを指定すれば，そのメンバは継承したクラスにおいてオーバライドできなくなる．

このようにクラスの使い方も，C++やJavaと近い考え方で利用できるため，それらの言語を利用したことがあれば，それほど苦労することはないだろう．



モジュールの利用と作成

Pikeでは「モジュール」を使って，機能を拡張することができる．標準でも，ネットワークプロトコルや画像操作，XMLパーサや暗号化処理などさまざまなモジュールが用意されており，高機能なプログラムを簡単に書ける(表2)．

モジュールを利用する際には，とくに宣言は必要なく，単にモジュール名と，そこに含まれるメンバの名前をピリオドで結んで記述するだけでよい．たとえば次の例は，「String」という文字列を操作するためのモジュールの「count」という文字列の中に含まれる部分文字列の数を数える関数を呼び出している．

```
int cnt = String.count("abcabcabc", "ab");
```

モジュールの中に，クラスが定義されている場合もある．次の例はファイルを読み込んで表示するサンプルだが，「Stdio」というモジュールで定義されている「File」というクラスを利用している．

```
Stdio.File f = Stdio.File();
if( f->open("test.txt","r") ){
  string dat = f->read();
  write( dat );
}
```

〔リスト3〕クラスの継承

```
class chihuahua
{
  inherit dog;
  void bark() {
    write("Yap yap!¥n");
  }
}
```

〔リスト4〕クラスのアクセス制御

```
class dog
{
  string name;
  private string last_food;
  void create(string n) {
    name = n;
  }
  void bark() {
    write("Bow wow!¥n");
  }
  final void eat(string food) {
    last_food = food;
    write("Yum yum...¥n");
  }
}
```

〔表2〕Pikeで用意されているモジュールの一部

Crypto	MD5やDES，RSAなどの暗号化/復号化の機能を提供
GTK	GTK+を利用したGUI機能を提供
Image	画像の加工や，さまざまな画像フォーマットの読み込みや書き出し，加工などの機能を提供
Mysql	MySQLへのアクセス機能を提供
PDF	PDFの書き出し機能を提供
Parser	XMLやHTML，SGMLなどを扱う機能を提供
Protocols	HTTP，SMTP，DNS，LDAPなど，さまざまなインターネットプロトコルを扱う機能を提供
Regexp	正規表現を利用する機能を提供
Web	ウェブロボットを利用できる機能を提供

モジュールの中に，さらにモジュールが定義されている場合もある．その場合は，すべてのモジュール名をピリオドでつなぐ．たとえば，次の場合は「Protocols」というインターネットプロトコル関係のモジュールを集めたモジュールの中に定義されている「HTTP」というモジュールのget_urlというメソッドを呼び出している．戻り値はやはり同じ「HTML」モジュール内で定義されている「Query」という変数型である．

```
Protocols.HTTP.Query url_data =
              Protocols.HTTP.get_url(
          "http://www.takaaki.info/")
```

importというキーワードを使うと，現在の名前空間を切り替えることができる．たとえば，前述の「Protocols.HTTP」の例をimportを使って書き直すと，次のようになる．

```
import Protocols.HTTP;
Query url_data = get_url(
             "http://www.takaaki.info/");
```

ただしimportは，Javaなどにおけるimport文と異なり，単に名前空間を切り替えるだけのものであり，プログラム中で

import文を何回も呼び出すことで，次々と利用する名前空間を切り替えることもできる．

```
import A;
some_method();
    // ↑ A.some_method()が呼び出される
import B;
some_method();
    // ↑ B.some_method()が呼び出される
```

さて，モジュールは自分で作成することもできる．一つのモジュールは,「.pmod」という拡張子をもつ一つのファイルで構成されている．Pikeのプログラムを「.pmod」という拡張子で保存するだけで，新しいモジュールを作成できる．作成したモジュールは，標準のモジュールファイルの保存場所だけでなく，実行する.pikeファイルと同じディレクトリに置くこともできる．その場合は,「.mymodule.function」のように，モジュール名の前にピリオドをもう一つ追加して指定する．

また，C言語で作成したプログラムを，モジュールとして利用することも可能になっている．



プリプロセッサ

Pikeでは，プログラムは一度バイトコードに変換されてから実行される．プログラムが実行されるとコンパイラによってバイトコードへの変換が行われるが，その前にプリプロセッサにより処理が行われる．プリプロセッサは，文字コードのチェックや，コメントの除去など，バイトコードへのコンパイルのための前処理を行うが，そのほかにもいくつかのディレクティブを識別することができる．

ディレクティブは「#」を先頭につけて表す．その書式は，ほとんどC/C++と同じである．#defineではシンボルや定義済みマクロなどを定義することができる．

```
#define TEST_VERSION
#define CYCLES 20
#define ROL(X,Y)
                    (((X)<<(Y))&7+((X)>>(8-(Y))))
```

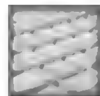
定義したシンボルを利用して，コンパイラに渡すコードを条件分岐させることもできる．

```
#if TEST_VERSION
  write ("This is test.¥n");
#endif
```

また，ユニークなのは#pikeというディレクティブで，これは，次のようにバージョンを指定することで，古いバージョンをエミュレートする機能を提供する．

```
#pike 7.0
```

自分の書いたプログラムが，どのバージョンなら正しく動作するのかといったことをチェックするには便利な機能といえるだろう．



エラーのハンドリングと実行速度の測定

Pikeにはcatchというキーワードが用意されており，これを利用すると，プログラムの実行時にエラーが発生した際に，処理を停止させることなく，発生したエラーメッセージを文字列として取得できるようになる．

```
int x,y;
array error = catch { x/=y; };
write( error[0]+".n" );
```

上記の例の場合，0で割り算をしようとしているため，「Division by zero.」という文字列が表示される．

さらにPikeには，catchとよく似た機能として，gaugeという機能も用意されている．こちらは，指定された処理を実行し，その実行速度を計測する．たとえば次のように使う．

```
int x, y=0;
float s = gauge {
  for( x = 0; x<1000000; x++ ){ y=100; }  };
write( "%f.n",s );
```

この場合，変数yに100という値を代入するという処理を100万回繰り返したときの処理速度が表示される．gaugeはプログラムを最適化する際や，処理ログに処理の所要時間を出力したい場合などには便利だろう．処理の速度を計測するという

〔リスト5〕GTKモジュールのサンプルプログラム

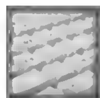
```
int main()
{
  GTK.setup_gtk();
  GTK.Button button = GTK.Button()->add(GTK.Label("Click Me!"));
  button->signal_connect("clicked", click);
  GTK.Window mainwindow = GTK.Window(GTK.WindowToplevel)-> set_policy(0,0,1);
  mainwindow-> signal_connect("destroy", exit, 0);
  mainwindow-> set_title("GTK TEST")
         -> add( button )
         -> show_all();
  return -1;
}

void click(){
    GTK.Alert("Hi!");
}
```

〔図2〕リスト5の実行結果



機能が実装されているというのは，なかなか面白い．

 GUIプログラミング

Pikeには，GUIライブラリのGTK+をパッケージ化した`GTK`というモジュールが標準で用意されており，これを利用すれば，GUIアプリケーションを作成することもできる．

GTKを利用したサンプルをリスト5に示す．このプログラムを実行すると，図2のようにボタンが一つだけついたウィンドウが表示され，ボタンをクリックすると，「Hi!」というメッセージが表示される．`GTK`モジュールには，Window，Buttonといった GUI のパーツがすべてクラスとして用意されており，プログラム中では，Windowを一つ作成し，そこにボタンを追加している．ボタンが押されたとき(clicked)には，`click`という関数が呼び出されるように設定し，clickでは`Alert`という関数で，メッセージボックスを表示している．

ちなみにWindows版でも，GTK+のダイナミックリンクライブラリが同梱されているため，同様にGUIプログラムを作成することが可能になっている．

おわりに

PikeはC/C++と非常によく似た言語仕様をもち，これらの言語を利用したことがあれば，非常にすんなりと入っていくことができるスクリプト言語である．さまざまな機能を提供するモジュールが用意されていて，高機能なプログラムも簡単に書くことができる．

ただ，Pikeはまだまだ非常にマイナな言語であることも事実だ．ドキュメントの整備も遅れており，情報が少ないことがもっとも大きな欠点となっている．リファレンスマニュアルも，ところどころ記述されていない(赤字で今後追加される，と書かれている)部分も目立つなど，まだまだ使いやすいとはいいづらい言語である．

しかし現在，PikeはRoxen Web ServerというWebサーバを記述するために開発が続けられている言語であり，そのWebサーバもきちんと公開されている．このことからも，実用に耐え得る言語であることは実証されているといってよいだろう．

世の中には，Perl，Ruby，Pythonといったもっとメジャーなスクリプト言語も数多く存在する．それらの言語はみな言語仕様が異なり，人によって好みがわかれるところだ．それらの言語がどうもしっくりこない，といった場合には，新しい選択肢としてPikeを考えてみるのもよいのではないだろうか．

みずの・たかあき

初級ドライバ開発者のための

Windowsデバイスドライバ開発テクニック

第4回　ReadFile(), WriteFile()への対応

丸山治雄

今回は，ReadFile()，WriteFile()に対応したドライバの作成方法を解説します．

4.1 即時処理モードとサブスレッド処理モード

ReadFile()とWriteFile()は，要求したアクセスが終了するまでアクセスを戻さない同期モードと，アクセスが終了したかどうかに関係なく制御をいったんアプリケーションに戻し，アクセスの終了を確認する必要が生じたときに確認を行う非同期モードがあります．

ドライバの作成方法としては，同期モードも非同期モードも違いはありませんが，ここではわかりやすさに重点を置くために，ドライバにリクエストが来たらアクセスを終了するまで制御を戻さない方式（即時処理モード）と，ドライバにアクセスが来たらアクセスの準備のみを行ってすぐにアプリケーションに処理を戻し，アクセス終了時にアプリケーションに通知する方式（サブスレッド処理モード）を説明します[注4.1]．

● アプリケーションの処理

KIT1050ボードのSRAMからデータを読み取るとき，アプリケーション側の記述としては，

```
ReadFile( hPCIHdl, DstBuff, DstSize,
                        &IOReturn, NULL );
```

のように，通常のアクセス要求を行います．

〔リスト4.1〕オーバラップモードでオープンしたときのアクセス要求

```
ovl.Offset = KIT1050DW0_OFFSET;  // 読み取りオフセットを必ず設定

if ( ReadFile( hPCIHdl, DstBuff, DstSize, &IOReturn, &ovl ) == TRUE )
{  // アクセス終了
   sts = IOReturn; // サイズを返す
}
else
{  // 転送終了を待つ
   if ( GetLastError() == ERROR_IO_PENDING )
   {
      GetOverlappedResult( hPCIHdl, &ovl, &IOReturn, TRUE );
      sts = IOReturn;     // サイズを返す
   }
```

注4.1：サブスレッドに処理を要求するモードは，KIT1050のソースリストには入っていない．

なお，ファイルをオープンするときにオーバラップモードでオープンしたときは，リスト4.1のようにアクセス要求を行います．

4.2 即時処理モード

前述したように，即時処理モードでは，ドライバにアクセス要求が来たときにドライバがアクセスを終了するまで，アプリケーションに制御を戻しません．動作の流れを図4.1に示します．

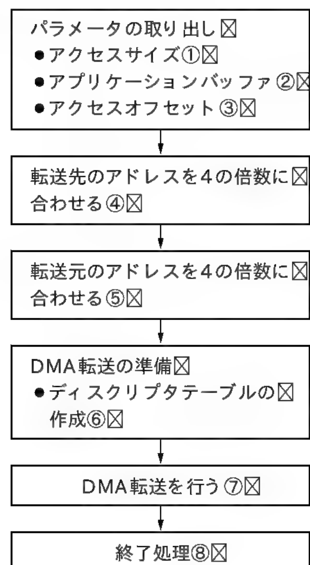
なお，ソースリストの解説は，リード要求に対応したもので説明します（リスト4.2）．

● パラメータの取り出し

ドライバは，リード要求，またはライト要求を受け付けると，アプリケーションからのアクセス要求が正しいかどうかを確認する必要があります．このチェックを省略すると，アプリケーションのReadFile()または，WriteFile()のパラメータに異常があったとき，システムが異常動作（いわゆる“ブルースクリーン”）を起こすおそれがあります．

まず，アプリケーションからのアクセス要求サイズを確認し

〔図4.1〕即時処理モードの動作



〔リスト 4.2〕即時処理モード

```
NTSTATUS
KIT1050Read(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
  PDEVICE_EXTENSION  pExtension = DeviceObject->DeviceExtension;
  PIO_STACK_LOCATION irpStack =
IoGetCurrentIrpStackLocation(Irp);
  PULONG  volatile RegAddr;
  DMA_LRB_REQUEST  DmaDescriptor;
  PDMA_DESCRIPTOR  DmaLink;
  LRB    SrcLRB;
  LRB    UnlockLRB;
  ULONG  DmaStatus;
  NTSTATUS status     = STATUS_SUCCESS;


  PUCHAR  pBuffer;           // 呼び出し元バッファのポインタ
  LARGE_INTEGER  rdOffset;   // 読み取り開始位置
  LONG  rdSize;              // 要求された読み込みサイズ
  LONG  rdBytesRead = 0;     // 読み込んだバイト長
  ULONG  offPos;


  rdSize = irpStack->Parameters.Read.Length;                    ┐
  if ( rdSize <= 0 )  // 読み込みサイズ0                          │
  {                                                             │
    status = STATUS_INVALID_PARAMETER;  // Driver Error         │
    Irp->IoStatus.Status       = status;                        ├①
    Irp->IoStatus.Information = 0;                              │
    IoCompleteRequest(Irp, IO_NO_INCREMENT);                    │
    return( status );                                           │
  }                                                             ┘

  // アプリケーションのバッファ                                  ┐
  pBuffer = Irp->MdlAddress;                                    │
  if ( pMdl == NULL )                                           │
  {                                                             │
    status = STATUS_INVALID_PARAMETER;  // Driver Error         ├②
    Irp->IoStatus.Status       = status;                        │
    Irp->IoStatus.Information = 0;                              │
    IoCompleteRequest(Irp, IO_NO_INCREMENT);                    │
    return( status );                                           │
  }                                                             ┘

  // 読み込みオフセット
  rdOffset = irpStack->Parameters.Read.ByteOffset;  ←── ③

  offPos = (ULONG)pBuffer & 3;                                  ┐
  if ( offPos != 0 )                                            │
                // 転送先のアドレスが4の倍数でないとき，サイズを調整  │
  {                                                             │
    offPos = 4 - offPos;                                        │
    rdTotalSize -= offPos;                                      │
    if ( rdTotalSize <= 0 )                                     │
    {                                                           ├④
      status = STATUS_END_OF_FILE;                              │
      rdBytesRead = 0;                                          │
      goto ReadExit;                                            │
    }                                                           │
  }                                                             │
  pBuffer = (PUCHAR)(((ULONG)pBuffer + 3) & 0xfffffffc);        │
                              // 転送先のアドレスは4の倍数          ┘

  // 領域内の確認                                                ┐
  offPos = rdOffset.LowPart;                                    │
  offPos &= 0xfffffffc;           // SRAMも4の倍数アドレス       │
  rdBytesRead = (offPos + rdTotalSize) - KIT1050MEMORY_SIZE;    │
  if ( rdBytesRead > 0 )                                        │
  {                                                             │
    rdTotalSize -= rdBytesRead;                                 │
    if ( rdTotalSize <= 0 )                                     ├⑤
    {                                                           │
      status = STATUS_END_OF_FILE;                              │
      rdBytesRead = 0;                                          │
      goto ReadExit;                                            │
    }                                                           │
    status = STATUS_END_OF_FILE;                                │
  }                                                             ┘
  rdTotalSize &= 0xfffffffc;
  if ( rdTotalSize <= 0 )
  {
    status = STATUS_END_OF_FILE;
    rdBytesRead = 0;
    goto ReadExit;
  }

  DmaDescriptor.PCILocalOffset = KIT1050MEMORY_OFFSET + offPos;  ┐
  DmaDescriptor.ChainMode = (PCI90X0_DIRECT_LOCAL_PCI |          │
                     PCI90X0_DESCRIPTOR_LOCAT);                  │
  DmaDescriptor.Descriptor = (PLRB)&SrcLRB;                      │
  DmaDescriptor.UnlockDescriptor = (PLRB)&UnlockLRB;             │
  SrcLRB.XferBuffer = pBuffer;                                   │
  SrcLRB.XferLength = rdTotalSize;                               │
                                                                 ├⑥
// DMA転送用ディスクリプタを作成します                            │
  offPos = MakeDMADescriptor( &DmaDescriptor, &SrcLRB );         │
    if (!NT_SUCCESS(offPos))                                     │
  {                                                              │
    status = offPos;                                             │
    rdBytesRead = 0;                                             │
    goto ReadExit;                                               │
  }                                                              ┘

  // DMAを初期化する                                              ┐
  DmaLink = (PDMA_DESCRIPTOR)SrcLRB.DescBuffer;                  │
  // 連続転送モード                                               │
  PCI9054RegPointer->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;       │
  PCI9054RegPointer->DMA0_MODE_REG |= (PCI90X0_DMA_CHAINING |    │
//                    PCI90X0_DMA_DONEINTENABLE |                │
                   PCI90X0_DMA_INTSELECT );                      │
  PCI9054RegPointer->DMA0_DESCRIPTOR_REG =                       │
                                 (ULONG)SrcLRB.DescAddress;      ├⑦
  PCI9054RegPointer->DMA0_DESCRIPTOR_REG |=                      │
                                 DmaDescriptor.ChainMode;        │
                                                                 │
  // DMA転送完了割り込みを許可                                    │
//  PCI9054RegPointer->SHARED_ICS |= PCI90X0_DMA0_INTENABLE;     │
                                                                 │
  // 転送開始                                                     │
  PCI9054RegPointer->DMA0_COMMAND_REG = PCI90X0_DMA_GO;          ┘

  // 転送終了を待つ                                               ┐
  RegAddr = (PULONG)&PCI9054RegPointer->DMA0_COMMAND_REG;        │
  while( 1 )                                                     │
  {                                                              ├⑧
    DmaStatus = (volatile)*RegAddr;                              │
    if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )                    │
      break;                                                     │
  }                                                              ┘

// 作成したディスクリプタを解放します
  ReleaseDMADescriptor( &SrcLRB );  ←── ⑨
  rdBytesRead = rdTotalSize;

ReadExit:
  Irp->IoStatus.Status = STATUS_SUCCESS;
  Irp->IoStatus.Information = 0;
  return( status );

}

NTSTATUS
KIT1050Write(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
  PDEVICE_EXTENSION  pExtension = DeviceObject->DeviceExtension;
  PIO_STACK_LOCATION irpStack =
                          IoGetCurrentIrpStackLocation(Irp);
  PULONG  volatile RegAddr;
  DMA_LRB_REQUEST  DmaDescriptor;
  PDMA_DESCRIPTOR  DmaLink;
  LRB    SrcLRB;
  LRB    UnlockLRB;
  ULONG  DmaStatus;
    NTSTATUS status     = STATUS_SUCCESS;


    PUCHAR  pBuffer;         // 呼び出し元バッファのポインタ
  LONG    wtSize;            // 要求された書き込みサイズ
  LARGE_INTEGER  wtOffset;   // 書き込み開始位置
```

〔リスト 4.2〕即時処理モード(つづき)

```
  LONG    wtBytesSent;        // 書き込んだバイト長
  ULONG   offPos;


  wtSize = irpStack->Parameters.Write.Length;
  if ( wtSize <= 0 )  // 読み込みサイズ0
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

  // アプリケーションのバッファ
  pBuffer = Irp->MdlAddress;
  if ( pMdl == NULL )
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

  // 読み込みオフセット
  wtOffset = irpStack->Parameters.Read.ByteOffset;

  offPos = (ULONG)pBuffer & 3;
  if ( offPos != 0 )
                  // 転送元のアドレスが4の倍数でないとき,サイズを調整
  {
    offPos = 4 - offPos;
    wtTotalSize -= offPos;
    if ( wtTotalSize <= 0 )
    {
      status = STATUS_END_OF_FILE;
      wtBytesSent = 0;
      goto WriteExit;
    }
  }
  pBuffer = (PUCHAR)(((ULONG)pBuffer + 3) & 0xfffffffc);
                       // 転送元のアドレスは4の倍数

  // 領域内の確認
  offPos = wtOffset.LowPart;
  offPos &= 0xfffffffc;           // SRAMも4の倍数アドレス
  wtBytesSent = (offPos + wtTotalSize) - KIT1050MEMORY_SIZE;
  if ( wtBytesSent > 0 )
  {
    wtTotalSize -= wtBytesSent;
    if ( wtTotalSize <= 0 )
    {
      status = STATUS_END_OF_FILE;
      wtBytesSent = 0;
      goto WriteExit;
    }
    status = STATUS_END_OF_FILE;
  }
  wtTotalSize &= 0xfffffffc;
  if ( wtTotalSize <= 0 )
  {
    status = STATUS_END_OF_FILE;
    wtBytesSent = 0;
    goto WriteExit;
  }

  DmaDescriptor.PCILocalOffset = KIT1050MEMORY_OFFSET + offPos;
  DmaDescriptor.ChainMode = (PCI90X0_DIRECT_PCI_LOCAL |
                PCI90X0_DESCRIPTOR_LOCAT);
  DmaDescriptor.Descriptor = (PLRB)&SrcLRB;
  DmaDescriptor.UnlockDescriptor = (PLRB)&UnlockLRB;
  SrcLRB.XferBuffer = pBuffer;
  SrcLRB.XferLength = wtTotalSize;

// DMA転送用ディスクリプタを作成します
  offPos = MakeDMADescriptor( &DmaDescriptor, &SrcLRB );
    if (!NT_SUCCESS(offPos))
  {
    status = offPos;
    wtBytesSent = 0;
    goto WriteExit;
  }

  // DMAを初期化する
  DmaLink = (PDMA_DESCRIPTOR)SrcLRB.DescBuffer;
  // 連続転送モード
  PCI9054RegPointer->DMA0_MODE_REG = PCI90X0_INI_DMA_MODE;
  PCI9054RegPointer->DMA0_MODE_REG |= (PCI90X0_DMA_CHAINING |
//                    PCI90X0_DMA_DONEINTENABLE |
                   PCI90X0_DMA_INTSELECT );
  PCI9054RegPointer->DMA0_DESCRIPTOR_REG =
(ULONG)SrcLRB.DescAddress;
  PCI9054RegPointer->DMA0_DESCRIPTOR_REG |=
DmaDescriptor.ChainMode;

  // DMA転送完了割り込みを許可
//  PCI9054RegPointer->SHARED_ICS |= PCI90X0_DMA0_INTENABLE;

  // 転送開始
  PCI9054RegPointer->DMA0_COMMAND_REG = PCI90X0_DMA_GO;

  // 転送終了を待つ
  RegAddr = (PULONG)&PCI9054RegPointer->DMA0_COMMAND_REG;
  while( 1 )
  {
    DmaStatus = (volatile)*RegAddr;
    if( (DmaStatus & PCI90X0_DMA_DONE) != 0 )
      break;
  }

// 作成したディスクリプタを解放します
  ReleaseDMADescriptor( &SrcLRB );
  wtBytesSent = wtTotalSize;

WriteExit:
  Irp->IoStatus.Status = STATUS_SUCCESS;
  Irp->IoStatus.Information = 0;
  return( status );
}
```

ます．アクセスサイズが0のときはアクセスできないので，ステータスを設定してすぐにリターンします(リスト4.2の①)．

次に，送受信バッファのMDLを取り出します．バッファアドレスは，アプリケーションの論理メモリポインタをドライバがアクセスできるようにアドレス変換し，ドライバにエントリします．もし，MDLがNULLのときはアクセスできないので，そのステータス(エラー)を設定してすぐにリターンします(リスト4.2の②)．

最後に，PCIのSRAMのアクセスオフセットを取り出します(リスト4.2の③)．ここで，アクセスオフセットをチェックしない理由は，後述するようにSRAMアドレスとセットでチェックを行っているためです．

● 転送サイズの調整

次に，PCIのメモリアクセスは4バイトバウンダリなので，送受信バッファと転送サイズが4の倍数であるかどうかをチェックし，もし4の倍数でないときは4の倍数になるようにバッファポインタとアクセスサイズを調整します．

まず，転送先(ReadFile()の場合．WriteFile()のときは転送元)のアプリケーションメモリポインタが4の倍数であるかをチェックし，4の倍数でないときは4の倍数になるようにMDLアドレスを切り上げます．

このとき，アクセスサイズが0以下になったときは，EOF

ステータスをアプリケーションに返し，すぐにリターンします(リスト4.2の④).

次に，PCIのSRAMアドレスが4の倍数であるかどうかをチェックします．読み取り開始アドレスは固定なので，オフセットの値のみチェックします．もし，読み取りサイズとオフセットを計算しSRAMサイズを超えているときは，アクセスサイズを調整します．調整の結果，アクセスサイズが0以下のときは，EOFステータスをアプリケーションに返してすぐにリターンします(リスト4.2の⑤).

● DMA転送の準備

次に，第3回で説明したDMA転送用のディスクリプタテーブルを作成します．`DmaDescriptor.PCILocalOffset`には，KIT1050のSRAMメモリの先頭アドレスにアプリケーションから指定されたオフセットを指定します．

`DmaDescriptor.ChainMode`には，PLX9054のDMA転送の方向とディスクリプタテーブルがPC側メモリにあることを指定します．

`DmaDescriptor.Descriptor`と`UnlockDescriptor`は，ディスクリプタテーブルを作成，解除するための構造体です．

`SrcLRB.XferBuffer`にアプリケーションのバッファMDLアドレス，`XferLength`にアクセスサイズを指定して，ディスクリプタテーブルを作成します(以上，リスト4.2の⑥).

● DMA転送と終了処理

作成したディスクリプタテーブルを使用してPLX9054のDMA転送用レジスタを設定します．この例では，第3回で説明した連続転送方式を使用してDMA転送を行っています(リスト4.2の⑦).

転送終了の確認で，レジスタ変数(`RegAddr`)を`volatile`宣言しているのは，毎回ステータスを読み取るためです．この宣言がないと，コンパイラがコードの最適化を行ってしまい，その結果，`while()`に入るときに1回だけレジスタアクセスを行うことになり，以後アクセスしません．したがって，必ずアクセスを行うために，`volatile`宣言が必要になります(リスト4.2の⑧).

転送が終了したら，DMA転送用に作成したディスクリプタを解放して(リスト4.2の⑨)，リード/ライトを終了します．

なお，この転送方式は，ドライバの中で転送終了を待つことになるので，ほかのドライバやアプリケーションの実行が一時的に停止することになります．

したがって，このサンプルを使用して大容量の転送を行うことはお勧めできません．あくまでも，小さなサイズのときにのみ行ってください．

4.3 サブスレッド処理モード

サブスレッド処理モードは，通常のドライバ作成においてよく使用される方式ですが，処理方法が複雑なので概要を説明します．動作の流れを図4.2に，プログラムをリスト4.3に示します．

〔図4.2〕サブスレッド処理モードの動作

● パラメータの取り出し

ドライバは，リード要求，またはライト要求を受け付けると，アプリケーションからのアクセス要求が正しいかどうかを確認する必要があります．このチェックを省略すると，アプリケーションの`ReadFile()`，または`WriteFile()`のパラメータに異常があったとき，システムが異常動作(いわゆる“ブルースクリーン”)を起こすおそれがあります．

まず，アプリケーションからのアクセス要求サイズを確認します，アクセスサイズが0のときはアクセスできないので，ステータスを設定してすぐにリターンします(リスト4.3の①).

次に，送受信バッファのMDLを取り出します．バッファアドレスは，アプリケーションの論理メモリポインタをドライバがアクセスできるようにアドレスを変換して，ドライバにエントリします．もし，MDLが`NULL`のときはアクセスできないので，そのステータス(エラー)を設定してすぐにリターンします(リスト4.3の②).

● サブスレッドの起動

ドライバはリード/ライトのリクエストを受け付けると，実際にアクセスを行うサブスレッドを起動します．

まず，サブスレッドの動作を開始するために，セマフォを初期化します(リスト4.3の③)．これは，たとえば割り込みが発生したときにアクセスを開始する場合に使用します(リスト4.4の①は，割り込みをトリガにするときに割り込み処理ルーチンの中に組み込むサンプルルーチン)．したがって，トリガがな

〔リスト 4.3〕サブスレッド 処理モード

```
//=============================================================
// KIT1050Read() - APからのReadFile()受け付け
//
// Parameters: DeviceObject - Device object for this
//             operation.
//             Irp - Irp to be cancelled.
// ============================================================
NTSTATUS
KIT1050Read(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
  PDEVICE_EXTENSION  pExtension =
                     DeviceObject->DeviceExtension;
  PIO_STACK_LOCATION irpStack =
                     IoGetCurrentIrpStackLocation(Irp);
  KIRQL    kCancelSpin;
  PMDL     pMdl;
  LONG     rdSize;
  LONG  Board;
  NTSTATUS  status = STATUS_PENDING;  // Driver status;


  // 読み込みサイズの取得
  rdSize = irpStack->Parameters.Read.Length;
  if ( rdSize <= 0 )  // 読み込みサイズ0
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;                    ①☒
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

  // アプリケーションのバッファ
  pMdl = Irp->MdlAddress;
  if ( pMdl == NULL )
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;                    ②☒
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

// アクセス用のスレッドを作成
    // Read Thread
    pExtension->ReadRun = TRUE;
    KeInitializeSemaphore(
          &pExtension->ReadSemaphoreObject,  ③☒
          0,
          1);
    status = PsCreateSystemThread(
          pExtension->ReadThreadHdl,
          THREAD_ALL_ACCESS,
          NULL,
          NULL,
          NULL,
          ReadRoutine,
          pExtension
          );
    if ( !NT_SUCCESS(status) )                 ④☒
    {
      pExtension->ReadRun = FALSE;
      KeInitializeSemaphore(   // Signal
          &pExtension->ReadSemaphoreObject,
          1,
          1);
    }

  // キャンセルルーチンの登録
  IoAcquireCancelSpinLock(&kCancelSpin);
  IoSetCancelRoutine(Irp, KIT1050ReadCancel); ⑤☒
  IoReleaseCancelSpinLock(kCancelSpin);
  // Irpをキューに登録
  ExInterlockedInsertTailList(&pExtension->ReadWaitQueue,
                  &Irp->Tail.Overlay.ListEntry,           ⑥☒
                  &pExtension->ReadQueueSpin);
  IoMarkIrpPending(Irp);

  Irp->IoStatus.Status = STATUS_SUCCESS;
  Irp->IoStatus.Information = 0;
  return( status );
}

//=============================================================
// ReadRoutine() - 割り込みが発生したときSRAMを読み出す
//
// Parameters: DeviceObject Extension - Device object for this
// operation.
// ============================================================
VOID ReadRoutine( IN PVOID Context )
{
  PDEVICE_EXTENSION  pExtension = (PDEVICE_EXTENSION)Context;
  PRiMAIN_REGISTER  RegPtr;
    PIO_STACK_LOCATION irpStack;
  KIRQL              kCancelSpin;
  PLIST_ENTRY        head;
  PIRP               NewIrp;
  PMDL       pMdl;
  PULONG       rdBuff;
  LARGE_INTEGER  rdOffset;
  ULONG       offPos;
  LONG       rdSize;
  LONG       Board;
  PULONG       FIFOPtr;
  NTSTATUS  status = STATUS_PENDING;  // Driver status;
  SRAMPtr = (PULONG)PCIinf.PCIMemAddr;


    // 割り込み待ち
    KeWaitForSingleObject(
        &pExtension->ReadSemaphoreObject,
        Executive,                            ⑦☒
        KernelMode,
        FALSE,
        0);

    // Closeしている
    if ( pExtension->ReadRun == FALSE ) ←── ⑧☒
    {
      KIT1050ReadAbort( pExtension );
      goto readEnd;
    }

    // Read要求が無い
    if (IsListEmpty(&pExtension->ReadWaitQueue))
    {  // RFRDYのクリア                             ⑨☒
      RegPtr->Ctrl0 = CTRL0_RFFCLR;
      continue;
    }

    IoAcquireCancelSpinLock(&kCancelSpin);
    head = ExInterlockedRemoveHeadList(
            &pExtension->ReadWaitQueue,
            &pExtension->ReadQueueSpin);
    NewIrp = CONTAINING_RECORD(                      ⑩☒
          head, IRP, Tail.Overlay.ListEntry);
    irpStack = IoGetCurrentIrpStackLocation(NewIrp);
    IoSetCancelRoutine (NewIrp, NULL);
    IoReleaseCancelSpinLock (kCancelSpin);

    rdSize = irpStack->Parameters.Read.Length;
    pMdl = NewIrp->MdlAddress;
    rdBuff = MmGetSystemAddressForMdl( pMdl );       ⑪☒
    rdOffset = irpStack->Parameters.Read.ByteOffset;
    offPos = rdOffset.LowPart;

    // SRAM読み取り
    RtlCopyMemory( (PVOID)rdBuff,
            (PVOID)SRAMPtr,
            rdSize
            );

    // ReadFile()要求元にステータスを返す
    NewIrp->IoStatus.Information = rdSize;
    NewIrp->IoStatus.Status = STATUS_SUCCESS;        ⑫☒
    IoCompleteRequest(NewIrp, IO_NO_INCREMENT);


readEnd:
  pExtension->ReadRun = FALSE;
  // Thread CleanUp                                  ⑰☒
```

〔リスト 4.3〕サブスレッド処理モード(つづき)

```
  PsTerminateSystemThread( STATUS_SUCCESS );
}

// ==========================================================
// KIT1050ReadAbort() - This function removes the irp from our
//                      queue and then calls to the support
//                      routines to cancel the irp.
//
// Parameters: DeviceObject - Device object for this
//             operation.
//             Irp - Irp to be cancelled.
// ==========================================================
VOID KIT1050ReadAbort(IN PDEVICE_EXTENSION pExtension )
{
  KIRQL              kCancelSpin;
  PLIST_ENTRY        head;
  PIRP               NewIrp;


  if ( IsListEmpty(&pExtension->ReadWaitQueue) ) ←──── ⑬⊠
    return;

  IoAcquireCancelSpinLock(&kCancelSpin);                    ┐
  head = ExInterlockedRemoveHeadList(                       │
                  &pExtension->ReadWaitQueue,               │ ⑭⊠
                  &pExtension->ReadQueueSpin);              │
  NewIrp = CONTAINING_RECORD(                               │
              head, IRP, Tail.Overlay.ListEntry);           ┘
  IoSetCancelRoutine (NewIrp, NULL);
  IoReleaseCancelSpinLock (kCancelSpin);              ┐
  NewIrp->IoStatus.Information = 0;                   │ ⑮⊠
  NewIrp->IoStatus.Status = STATUS_CANCELLED;         │
  IoCompleteRequest(NewIrp, IO_NO_INCREMENT);         ┘
}

// ==========================================================
// KIT1050ReadCancel() - This function removes the irp from
//                     our queue and then calls to the support
//                     routines to cancel the irp.
//
// Parameters: DeviceObject - Device object for this
//             operation.
//             Irp - Irp to be cancelled.
// ==========================================================
VOID KIT1050ReadCancel(
             IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
  KIRQL kOld;
  PDEVICE_EXTENSION  pExtension =
             DeviceObject->DeviceExtension;

  KeAcquireSpinLock( &pExtension->ReadQueueSpin, &kOld );  ┐
  RemoveEntryList( &Irp->Tail.Overlay.ListEntry );         │ ⑯⊠
  KeReleaseSpinLock( &pExtension->ReadQueueSpin, kOld );   │
  IoReleaseCancelSpinLock( Irp->CancelIrql );              ┘

  Irp->IoStatus.Status = STATUS_CANCELLED;
  Irp->IoStatus.Information = 0;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);

}

// ==========================================================
// KIT1050Write() - APからのWriteFile()受け付け
//
// Parameters: DeviceObject - Device object for this
//             operation.
//             Irp - Irp to be cancelled.
// ==========================================================
NTSTATUS
KIT1050Write(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    )
{
  PDEVICE_EXTENSION  pExtension=
              DeviceObject->DeviceExtension;
  PRiMAIN_REGISTER  RegPtr;
  PIO_STACK_LOCATION irpStack =
              IoGetCurrentIrpStackLocation(Irp);
  KIRQL    kCancelSpin;
  PMDL    pMdl;
  LONG    wtSize;
  LONG  Board;
  NTSTATUS  status = STATUS_PENDING;  // Driver status;


  Board = pExtension->Board;
  RegPtr = PCIRegPointer[Board];

  wtSize = irpStack->Parameters.Write.Length;
  if ( wtSize <= 0 )  // 書き込みサイズ0
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

  pMdl = Irp->MdlAddress;
  if ( pMdl == NULL )
  {
    status = STATUS_INVALID_PARAMETER;  // Driver Error
    Irp->IoStatus.Status      = status;
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return( status );
  }

// アクセス用のスレッドを作成
    // Write Thread
    pExtension->WriteRun = TRUE;
    KeInitializeSemaphore(
          &pExtension->WriteSemaphoreObject,
          0,
          1);
    status = PsCreateSystemThread(
          pExtension->WriteThreadHdl,
          THREAD_ALL_ACCESS,
          NULL,
          NULL,
          NULL,
          WriteRoutine,
          pExtension
          );
    if ( !NT_SUCCESS(status) )
    {
      pExtension->WriteRun = FALSE;
      KeInitializeSemaphore(   // Signal
          &pExtension->WriteSemaphoreObject,
          1,
          1);
    }

  // キャンセルルーチンの登録
  IoAcquireCancelSpinLock(&kCancelSpin);
  IoSetCancelRoutine(Irp, KIT1050WriteCancel);
  IoReleaseCancelSpinLock(kCancelSpin);
  // Irpをキューに登録
  ExInterlockedInsertTailList(&pExtension->WriteWaitQueue,
              &Irp->Tail.Overlay.ListEntry,
              &pExtension->WriteQueueSpin);
  IoMarkIrpPending(Irp);

  Irp->IoStatus.Status = STATUS_SUCCESS;
  Irp->IoStatus.Information = 0;
  return( status );
}
// ==========================================================
// WriteRoutine() - 割り込みが発生したときSRAMに書き込む
//
// Parameters: DeviceObject Extension - Device object for this
//  operation.
// ==========================================================
VOID WriteRoutine( IN PVOID Context )
{
  PDEVICE_EXTENSION  pExtension = (PDEVICE_EXTENSION)Context;
  PRiMAIN_REGISTER  RegPtr;
   PIO_STACK_LOCATION irpStack;
  KIRQL              kCancelSpin;
  PLIST_ENTRY        head;
  PIRP               NewIrp;
  PMDL    pMdl;
```

〔リスト4.3〕サブスレッド処理モード(つづき)

```
    PULONG    wtBuff;
    LONG    wtSize;
    LONG  Board;
    PULONG  SRAMPtr = (PULONG)PCIinf.PCIMemAddr;
                                      // KIT1050のSRAM
    NTSTATUS  status = STATUS_PENDING;  // Driver status;


      // 割り込み待ち
      KeWaitForSingleObject(
          &pExtension->WriteSemaphoreObject,
          Executive,
          KernelMode,
          FALSE,
          0);

      // Closeしている
      if ( pExtension->WriteRun == FALSE )
        goto writeEnd;

      // Read要求が無い
      if (IsListEmpty(&pExtension->WriteWaitQueue))
        continue;

      IoAcquireCancelSpinLock(&kCancelSpin);
      head = ExInterlockedRemoveHeadList(
               &pExtension->WriteWaitQueue,
               &pExtension->WriteQueueSpin);
      NewIrp = CONTAINING_RECORD(
                head, IRP, Tail.Overlay.ListEntry);
      irpStack = IoGetCurrentIrpStackLocation(NewIrp);
      IoSetCancelRoutine (NewIrp, NULL);
      IoReleaseCancelSpinLock (kCancelSpin);

      wtSize = irpStack->Parameters.Write.Length;
      pMdl = NewIrp->MdlAddress;
      wtBuff = MmGetSystemAddressForMdl( pMdl );

      // SRAMに書き込み
      RtlCopyMemory( (PVOID)SRAMPtr,
              (PVOID)wtBuff,
              wtSize
                );

      // WriteFile()の要求元にステータスを返す
      NewIrp->IoStatus.Information = wtSize;
      NewIrp->IoStatus.Status = STATUS_SUCCESS;
      IoCompleteRequest(NewIrp, IO_NO_INCREMENT);

writeEnd:
  pExtension->WriteRun = FALSE;
  // Thread CleaneUp
  PsTerminateSystemThread( STATUS_SUCCESS );
}

// ==============================================================
// KIT1050WriteCancel() - This function removes the irp from
//                 our queue and then
//                 calls to the support routines to cancel
//                 the irp.
//
// Parameters: DeviceObject - Device object for this
//             operation.
//             Irp - Irp to be cancelled.
// ==============================================================
VOID KIT1050WriteCancel(
          IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
  KIRQL kOld;
  PDEVICE_EXTENSION  pExtension =
                       DeviceObject->DeviceExtension;


  KeAcquireSpinLock( &pExtension->WriteQueueSpin, &kOld );
  RemoveEntryList( &Irp->Tail.Overlay.ListEntry );
  KeReleaseSpinLock( &pExtension->WriteQueueSpin, kOld );
  IoReleaseCancelSpinLock( Irp->CancelIrql );

  Irp->IoStatus.Status = STATUS_CANCELLED;
  Irp->IoStatus.Information = 0;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);

}
```

く，すぐにアクセス可能なデバイスのときは必要ありません．

次に，サブスレッドを起動します(リスト4.3の④)．このとき，アクセスが取り消されたときのために，キャンセルルーチンも登録します(リスト4.3の⑤)．

最後にIrpを実行待ちキューに登録し，Irpをペンディングにします(リスト4.3の⑥)．ドライバは，アクセスを行わずにアプリケーションに制御を戻します．このとき，アプリケーションはERROR_IO_PENDINGを受け取ります．

● サブスレッドの動作

サブスレッドが動作してウェイトが解除されると，アクセスの準備に入ります．先ほども述べましたが，ウェイトが必要なときにのみ使用してください(リスト4.3の⑦)．

動作を開始すると，まずアプリケーションが動作しているかどうかを確認します．アプリケーションが終了しているときは，アクセスを行わずにサブスレッドを終了します．リスト4.3の⑧で，ドライバ内変数ReadRunはアプリケーションが取り消されるとFALSEになるように設計されています．

さらに，わずらわしいですが，Irpがキューに登録されているか否かを確認します．もし，登録されていないとき(アクセスがキャンセルされているとき)は，アクセスを行わずにサブスレッドを終了します(リスト4.3の⑨)．

アクセス開始のために，まずリクエストされたときのIrpを取り出します(リスト4.3の⑩)．Irpを取り出すと同時に，キャンセルルーチンも取り消します．取り出したIrpから，アクセスサイズ，バッファのMDLアドレス，オフセットを取り出します(リスト4.3の⑪)．

なお，このサンプルでは行っていませんが，即時処理モードで説明したとおり，リクエストが適正かどうかのチェックは必ず行ってください．このチェックはサブスレッドを起動する前に行ってもかまいません．

このサンプルでは，PCIメモリとPCのメモリとの転送は通常のメモリ転送を使用していますが，もちろんDMA転送で行っても問題ありません．

転送終了後にアクセスサイズをアプリケーションに戻します(リスト4.3の⑫)．最後にスレッドを解除して，一連のアクセス要求は終了します．

● サブスレッドに必要な処理関数

サブスレッドを起動したときに必要な処理関数は，以下のとおりです．

アボート処理

```
KIT1050ReadAbort(IN PDEVICE_EXTENSION
    pExtension )
```

PCICHECKのじょうずな使いかた

Column

第3回のコラム「DOSエクステンダでハードウェアデバッグ」で紹介したPCICHECKの便利な機能について解説します．

● 起動オプション

PCICHECKを起動するとき，/vオプションで表示するPCIボードを指定することができます．/vに続けて，デバイスIDとベンダIDを4桁の16進数で指定します．

KIT1050を指定するときは，以下のように指定します．

```
PCICHECK  /v002213d6
```

● デバイスIDを追加する方法

Device Name欄に表示するデバイス名を登録するときは，インターネットなどからvendors.txtを入手し，次のようにファイルに追加します．

```
13D6<TAB>K.I. Technology Co Ltd
<TAB>0017<TAB>KIT1030 PLX Getter
<TAB>0022<TAB>KIT1050 PLX Getter II
13D7<TAB>Toshiba Engineering Corporation
```

● マクロファイルの使用方法

PCICHECKは，マクロを実行することができます．マクロファイルの一例をリスト4.Aに示します．マクロ定義の詳細は，PCICHECKのアーカイブファイルに含まれるMacro.txtを参考にしてください．実行はステップのみです．ステップ実行するときは，F8キーを押してください．

〔リスト4.A〕マクロファイルの内容

```
; KIT1050 PLX Getter Ⅱ マクロサンプル

; ベンダー定義
Vendor=13d6¥0022  // PCIボードを指定します．

; PLX9054 Local Config.の定義
Base = 0
RegText  //「Register」で表示されるコメントの定義開始
    68=ICS
RegEnd   // RegTextの定義終了

; KIT1050の定義
Base = 2 // Base Addr. [2]の番号を指定
RegText  //「Register」で表示されるコメントの定義開始
    00=Status,INT
    04=Control_0,BRST,INTC
    08=Control_1,INTE
    0c=DipSW,SW-0,SW-1,SW-2,SW-3
    10=LED,LED-0,LED-1,LED-2,LED-3
    14=PIO-0
    18=PIO-1
    1c=PIO-CTRL,IOC0,IOC1
RegEnd   // RegTextの定義終了

Base = 2 // Base Addr. [2]の番号を指定
Mask
    //「Register」でアクセスするMask定義開始
    Status=R,01
    Control_0=W,03
    Control_1=RW,01
    0c=R,0f
    LED=RW,0f
    14=RW,ff
    18=RW,ff
    1c=RW,03
MaskEnd  // Maskの定義終了

Main
 Base=2
 LED = 0x00
 While LED == 0x00
    if DipSW == 8
         10 ^= 05  // 直接設定も可能
    else
         LED = 0x0f
    ifend
 wend
End

IntProc    // 割り込みの処理を定義
 Base = 0  // PLXの割り込みを許可
  ICS |= 800
 Base = 2
  08 |= 1  //Ctrl-1 Inte=On
 Wait
 Base = 2
  Status Status==1
  Read=0c  // Dip SW
  Read=LED // LED
  IntClr Control_0 = 2
  Set LED ^= 5
 Base = 0  // PLXの割り込みを禁止
  ICS &= ~800
 Base = 2
  08 &=~ 1
IntEnd     // IntProcの定義終了
```

キャンセル処理

VOID KIT1050ReadCancel(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)

アボート処理は，キューに保存されているIrpを取り消す処理を行います．まず，キューに保存されているIrpがあるか否かをチェックします（リスト4.3の⑬）．

保存されているIrpがないときは，何も行わずにリターンします．保存されているIrpがあるときは，そのIrpを取り出し（リスト4.3の⑭），アプリケーションにアクセスが取り消されたことを通知します（リスト4.3の⑮）．

キャンセル処理は，アクセス処理そのものを取り消します（リスト4.3の⑯）．ただし，サブスレッドを取り消さないので，リスト4.4の②の例のように，アプリケーションがキャンセルされたときにサブスレッドが起動しているときは，強制的にサブスレッドを取り消す必要があります．

〔リスト4.4〕強制的にサブスレッドを取り消す

```
// Read/Writeのサブスレッドが割り込みのとき，
// 割り込みルーチン内で，セマフォをオンにする

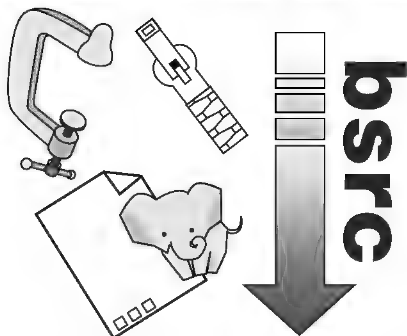
if ( pExtension->ReadRun == TRUE )                    ┐
{  // ReadRoutine()をアクティブにする                  │
  KeReleaseSemaphore(                                  │
      &pExtension->ReadSemaphoreObject,                ├ ①⊠
      IO_NO_INCREMENT,1,                               │
      FALSE);                                          │
}                                                      ┘

// アクセススレッドの解除例
// IRP_MJ_CLOSEの中での解除

    case IRP_MJ_CLOSE:                                 ┐
                                                       │
             ⋮                                         │
                                                       │
if ( pExtension->ReadRun == TRUE )                     │
{                                                      │
  pExtension->ReadRun = FALSE;                         ├ ②⊠
  KeReleaseSemaphore(                                  │
      &pExtension->ReadSemaphoreObject,                │
      IO_NO_INCREMENT,1,                               │
      FALSE);                                          │
}                                                      ┘

             ⋮
```

まるやま・はるお　ドライバ屋



ブロックソートとレンジコーダによるファイルの圧縮

高性能圧縮ツールbsrcの理論と実装（後編）

広井 誠

現在までに，さまざまな汎用ファイル圧縮ツールが公開されている．前号に引き続き解説を行うbsrcは，圧縮アルゴリズムとしてブロックソートとレンジコーダを採用し，高圧縮率で知られ，Linuxカーネルソースの配布などでも用いられるbzip2並の圧縮率を実現する．

後編となる今回は，bsrcで使われているレンジコーダをはじめとしたデータの符号化について解説する．なお，bsrcの全ソースリストおよび実行ファイルはWeb上で公開されているほか，本誌付属CD-ROMへも収録した．（編集部）

前回は，データを圧縮する前に，「圧縮しやすい形式」へとデータを変換するブロックソートとMTF法，そして基本的な圧縮アルゴリズムであるランレングスについて解説を行いました．

今回は圧縮処理の柱である，符号化について解説を行います．

レンジコーダとはなにか

ハフマン符号と算術符号は，記号の出現確率だけを利用してデータを圧縮する方法です．算術符号はハフマン符号よりも性能が良いのですが，実現方法が難しくて実行速度がハフマン符号よりも遅く，なおかつ特許の問題もあって，一般にはあまり普及していないのが現状です．

ところが，最近になってレンジコーダ（RangeCoder）という方法が注目されています．レンジコーダは原理的には算術符号と同じ方法ですが，参考文献3）によると『（おそらく）特許フリー』とのことで，性能は算術符号に比べるとわずかに劣りますが，実現方法はとても簡単で実行速度も高速です．もちろん，ハフマン符号よりも高性能です．ブロックソートでファイルを圧縮する場合，ハフマン符号の代わりにレンジコーダを使用することで圧縮率を改善することができます．

算術符号

● 算術符号の符号化

レンジコーダについて説明する前に，まず算術符号の基本的な考え方を説明します．算術符号は記号列全体を一つの符号語にする方法で，1960年代にP.Eliasによって提案されました．

算術符号は記号列を実数0と1の間の区間を用いて表します．たとえば，記号は{a，b，c}の3種類があり，出現確率はそれぞれ0.2，0.6，0.2とします．算術符号は，区間を記号の出現確率に比例した小区間に分割していくことで符号化を行います．それでは，記号列abbbcを符号化してみましょう．

図1を見てください．x以上y未満の区間を[x，y)と表すことにします．区間の初期値は[0，1)です．記号を読み込んだら区間[0，1)を分割します．記号がaならば区間の0から0.2までの部分，bならば0.2から0.8までの部分，cならば0.8から1.0までの部分に分割します．

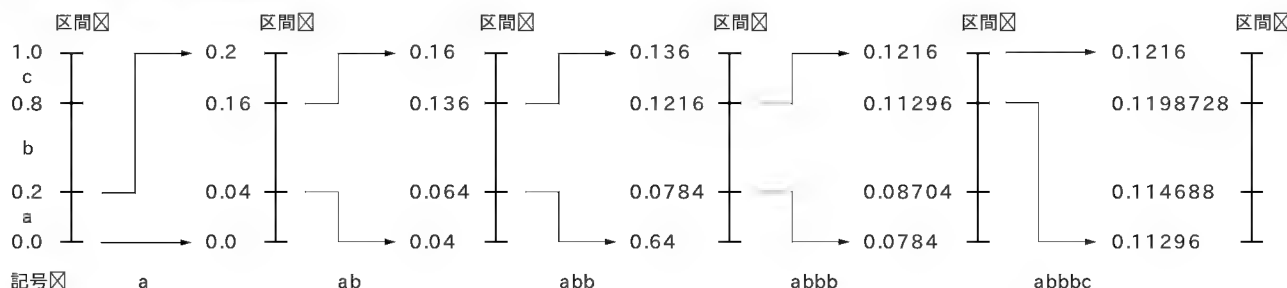
最初の記号はaなので区間は[0，0.2)となります．次の記号はbなので，区間[0，0.2)の0.2から0.8までの部分[0.04，0.16)が新しい区間になります．このように，記号を読み込むたびに区間を分割していくと，記号列abbbcを表す区間は[0.11296，0.1216)になります．そして実際の符号語は，この区間に含まれる一つの実数を指定します．

ここで符号語を2進数で表して，区間内で小数点以下のビット数の少ない値を選ぶことにします．たとえば，0.1171875を2進数で表すと次のようになります．

$$0.1171875 = 1/16 + 1/32 + 1/64 + 1/128 = (0.0001111)$$
（2進数）

0001111の7ビットを符号語として出力すれば，記号列abbbcの5文字を7ビットに圧縮することができます．この例では1文字が1.4ビットに圧縮されましたが，記号の出現確率によっては1文字が1ビット未満に圧縮できる場合もあります．ちな

〔図1〕符号化の過程（算術符号）



みに，ハフマン符号では1文字が1ビット未満になることはありえません．

● 算術符号の復号

次は復号を説明します．ここでは説明の都合上，符号語は区間の下限値0.11296とします．0.11296は[0, 0.2)の間にあるので，最初の記号はaであることがわかります．次に，aを表す区間[0, 0.2)を[0, 1.0)になるように拡大すると，符号語は次のように変換できます．

新しい符号語 =(符号語－記号の下限値)/記号の区間幅
=(0.11296－0)/0.2
=0.5648

新しい符号語0.5648は[0.2, 0.8)の間にあるので，次の記号はbであることがわかります．このような操作を繰り返し行うことで，表1のように記号列abbbcを復号することができます．

ところで，算術符号には二つの問題点があります．一つは記号列の最後を判定できないことです．さきほどの復号の例では最後に符号語が0になりましたが，このあとも記号aを復号することができます．つまり，符号語0.11296は記号列abbbcだけではなく，abbbca，abbbcaa，abbcaaa……などにも復号することができるのです．この問題は，終端を表す記号を用意して終端記号を復号したら終了する，または，記号の総数をファイルの先頭に書き込んでおく，などといった方法で解決することができます．

もう一つは，入力する記号列が長くなるほど，より多くの桁数が必要になることです．また，浮動小数点演算の誤差も考慮しなければいけません．これはとても大きな問題点で，解決するまでに10年以上の時間がかかりました．1981年にC.B.Jonesによって発表された算術符号(Jones符号)は，実数のかわりに整数で演算するように工夫されています．算術符号に興味のある方は参考文献1)，参考文献2)を参照してください．



レンジコーダ

● レンジコーダの基本的な考え方

レンジコーダは1998年にMichael Schindler[9]が発表し，「高性能，高速，特許フリー」の方法として注目を集めるようになりました．Michael Schindlerのレンジコーダは計算の途中で「桁上がり」が発生しますが，ロシアのDmitry Subbotinが発表した「桁上げのないレンジコーダ」は，その名のごとく桁上がりが発生しません．現在，レンジコーダはおもにこの2種類の形式が存在するようです．

それでは，レンジコーダの基本的な考え方について説明します．ここで説明するレンジコーダは「桁上がり」が発生するバージョンです．「桁上げのないレンジコーダ」は参考文献[3]か筆者のWebページ[11]を参照してください．

算術符号は区間[0, 1)を分割していきますが，レンジコーダは[0, 1)を分割するのではなく，最初に大きな区間，たとえば

〔表1〕復号の過程(算術符号)

符号語	記　号	区　間
0.11296	a	[0, 0.2)
0.5648	b	[0.2, 0.8)
0.608	b	[0.2, 0.8)
0.68	b	[0.2, 0.8)
0.8	c	[0.8, 1)
0		

[0, 1000)を設定して，それを小さな区間に分割していくことで符号化を行います．レンジコーダは整数で演算するので，記号列が長くなると当然ですが区間が狭くなって分割できなくなります．そのときは区間を伸張することで対応します．

たとえば，[0, 1000)を分割していくと[123, 124)になりました．もうこれ以上分割できないので，区間をたとえば100倍して[12300, 12400)を分割することにします．このとき，区間全体の大きさは[0, 1000)ではなく，それを1000倍した[0, 100000)と考えるわけです．

単純に考えると，区間を表すために多倍長整数が必要になりますが，区間を引き伸ばすタイミングを定めることにより，通常の整数演算でレンジコーダをプログラムすることができます．また，区間全体の大きさも覚えておく必要はありません．レンジコーダは分割した区間の幅(range)と下限値だけで符号化することができます．復号の処理でも，符号化と同じタイミングでrangeを伸張していくことで，符号語を記号列に復号することができます．

● レンジコーダの符号化

それでは具体的にレンジコーダの符号化を説明します．ここでは説明の都合上，区間の幅rangeを0x1000000に設定します．実際にはrangeを0xffffffff(32ビット)に設定する場合が多いでしょう．下限値lowは0に初期化します．区間は[low, low＋range)と表すことができるので，最初の区間は[0, 0x1000000)となります．また，rangeの初期値が0x1000000なので，lowの値は0から0xffffffまでの範囲(24ビット)になります．

記号の出現確率により区間を分割するところは算術符号と同じです．レンジコーダはrangeが一定の値より小さくなった時点で，rangeを引き伸ばすところがポイントです．レンジコーダでは，rangeが初期値の1/256(0x10000)より小さくなったら256倍します．これは処理をバイト単位で行うための工夫です．次の例を見てください．

[0x123456, 0x123456＋0xabcd)＝　256倍
=> [0x12345600, 0x12345600＋0xabcd00)

いまlowの値が0x123456でrangeの値が0xabcdだとします．0xabcd<0x10000なのでrangeを256倍します．このとき，lowの値もいっしょに256倍します．これで区間を正しく表すことはできますが，このままではlowの値が大きくなる一方です．そこで，lowの値を一定の範囲内(24ビット)に収めることを考えます．

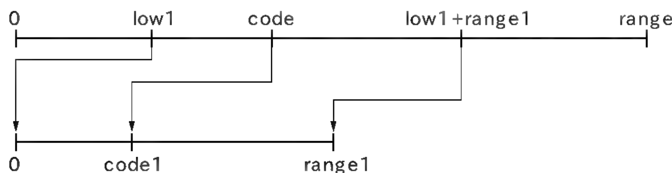
〔図2〕符号化の過程(レンジコーダ)

```
low   = low +( range × 記号の下限値) ☒
range = range × 記号の出現確率☒

[       low,     range]      [       low,    range]( 数値は 16 進数) ☒
[         0,1000000]—d→[ e00000, 200000] ☒
[ e00000, 200000]—c→[ f80000 ,  40000] ☒
[ f80000 ,  40000]—b→[ fa0000 ,  10000] ☒
[ fa0000 ,  10000]—b→[ fa8000 ,   4000] 256 倍して fa を出力☒
[ 800000, 400000]—a→[ 800000, 200000] ☒
[ 800000, 200000]  a →[ 800000, 100000] ☒
[ 800000, 100000]  a →[ 800000,  80000] ☒
[ 800000,  80000]—a→[ 800000,  40000] low[ 80, 00, 00]を出力☒

符号語 =>[ fa, 80, 00, 00] ☒
```

〔図3〕区間の更新(レンジコーダ)



〔図4〕復号の過程(レンジコーダ)

```
code  = code -( range × 記号の下限値) ☒
range = range × 記号の出現確率☒

符号語を 3 バイト[ fa, 80, 00] 読み込み code を初期化☒

[     code,     range]      [     code,     range]( 数値は 16 進数) ☒
[  fa8000, 1000000]—d→[ 1a8000,  200000] ☒
[ 1a8000,  200000]—c→[  28000,   40000] ☒
[   28000,   40000]  b →[   8000,   10000] ☒
[    8000,   10000]  b →[       0,    4000] 256 倍する☒
                                         符号語を 1 バイト( 00)code に加算 ☒
[        0,  400000]—a→[       0,  200000] ☒
[        0,  200000]—a→[       0,  100000] ☒
[        0,  100000]—a→[       0,   80000] ☒
[        0,   80000]—a→[       0,   40000] ☒

記号列 => "dcbbaaaa"
```

〔表2〕記号の出現確率

記　号	a	b	c	d
出現確率	1/2	1/4	1/8	1/8
下限値	0	4/8	6/8	7/8
上限値	4/8	6/8	7/8	8/8

rangeの値は24ビットの範囲内に収まるので，lowの計算は24ビットの足し算になります．桁上がりの処理を工夫すれば，lowを24ビットで保持することが可能です．たとえば，次のようにlowの上位8ビット(0x12)をバッファへ出力します．

[0x12345600, 0x12345600＋0xabcd00)
=> [0x345600, 0x345600＋0xabcd00)
low(0x12345600)の上位8ビット(0x12)をバッファへ出力
=>(0x12)

値をバッファに溜めておけば，桁上がりには簡単に対応することができます．また，桁上がりが発生しないように工夫することができれば，上位8ビット(0x12)をそのまま符号語としてファイルへ出力することができます．あとは，記号を読み込んで区間の分割と伸張を繰り返して，最後にlowの値(24ビット)を出力します．

簡単な例を示しましょう．記号列“dcbbaaaa”を符号化します．符号化の過程を図2に，記号の出現確率を表2に示します．

記号を読み込むたびに，rangeの値は小さくなりlowの値は増えていきます．d，c，b，bまで記号を読み込むと，rangeは0x4000になり0x10000より小さくなります．ここでrangeとlowを256倍して，lowの上位8ビット(0xfa)を出力します．次に記号aを読み込みます．rangeの値は小さくなりますが，aの下限値が0なのでlowの値は増えません．最後にlowの値を出力して終了です．符号語は[0xfa, 0x80, 0, 0]になります．

●レンジコーダの復号

次は復号について説明します．下限値lowと幅rangeは符号化と同様に0と0x1000000に初期化します．符号語をcodeとすると，最初lowは0なので[0, range)の範囲でcodeに対応する記号を探すことになります．見つけた記号をc1とすると，lowとrangeの値は符号化と同様に次式で更新します．

low1＝low(0)+(range×記号c1の下限値)
range1＝range× 記号c1の出現確率

今度は[low1, low1＋range1)の範囲でcodeに対応する記号を探します．ここでcodeから下限値の増分を引き算した値code1を求めます．すると，図3に示すようにcode1は区間[0, range1)の符号語に対応していることがわかります．つまり，次は[0, range1)の範囲でcode1に対応する記号を探せばよいのです．

このように，符号語codeから下限値の増分を引き算することで，区間を[low, low＋range)から[0, range)に変換することができます．したがって，復号処理では下限値lowの値を覚えておく必要はありません．

rangeが0x10000より小さくなったらrangeを256倍するのは符号化と同じです．このとき符号語codeも256倍して，新しい符号語を1バイト読み込んでcodeに加算します．これで符

〔リスト 1〕レンジコーダの符号化と復号

```
/* マクロ定義 */
#define MAX_RANGE   0x01000000
#define MIN_RANGE   0x00010000
#define MASK        0x00ffffff
#define CODE_SIZE   257
#define END         256

/* 出現頻度表 */
int count[CODE_SIZE];
int count_sum[CODE_SIZE + 1];

/* 出現頻度表の作成 */
void make_count_table( Uchar *buff, int size )
{
  int i;
  for( i = 0; i < CODE_SIZE; i++ ) count[i] = 0;
  for( i = 0; i < size; i++ ) ++count[ *buff++ ];
  count[END] = 1;
  count_sum[0] = 0;
  for( i = 0; i < CODE_SIZE; i++ ){
    count_sum[i + 1] = count_sum[i] + count[i];
  }
}

/* 桁上がりの処理 */
void overflow( Uchar *out, int wp )
{
  int i;
  for( i = wp - 1; i >= 0; i-- ){
    int code = out[i] + 1;
    out[i] = code & 0xff;
    if( code < 256 ) break;
  }
}

/* 符号化 */
int range_encode( Uchar *out, Uchar *in, int size )
{
  int i, wp = 0, range = MAX_RANGE, low = 0;

  for( i = 0; i <= size; i++ ){
    int tmp = range / count_sum[CODE_SIZE];
    int c = (i < size ? *in++ : END);

    low += tmp * count_sum[c];    /* range と low の計算 */
    range = tmp * count[c];

    if( low >= MAX_RANGE ){      /* 桁上がりのチェック */
      overflow( out, wp );
      low -= MAX_RANGE;
    }

    while( range < MIN_RANGE ){     /* range の拡張 */
      out[wp++] = low >> 16;
      low = (low << 8) & MASK;
      range <<= 8;
    }
  }
  /* 最後の出力 */
  out[wp++] = low >> 16;
  out[wp++] = (low >> 8) & 0xff;
  out[wp++] = low & 0xff;
  return wp;
}

/* 復号 */
int range_decode( Uchar *out, Uchar *in )
{
  int wp = 0, range = MAX_RANGE, code;
  code = *in++;
  code = (code << 8) + *in++;
  code = (code << 8) + *in++;

  while( 1 ){
    int c;
    int tmp  = range / count_sum[CODE_SIZE];
    int tmp1 = code / tmp;

    /* とりあえず単純な線形探索 */
    for( c = 0; c < CODE_SIZE; c++ ){
      if( (count_sum[c] <= tmp1) && (tmp1 < count_sum[c + 1]) )
                                                           break;
    }
    if( c == END ) break;           /* 終了 */
    out[wp++] = c;                  /* 出力 */

    code -= count_sum[c] * tmp;     /* range と code の計算 */
    range = count[c] * tmp;

    while( range < MIN_RANGE ){     /* range の拡張 */
      range <<= 8;
      code = ((code << 8) & MASK) + *in++;
    }
  }
  return wp;
}
```

号語を復号することができます．

それでは，復号の過程を具体的に説明します．図4を見てください．

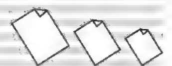
最初にrangeとcodeを初期化します．codeの範囲は24ビットなので，3バイト読み込んで0xfa8000に初期化します．次に,「記号の下限値 <= code/range < 記号の上限値」を満たす記号を探します．この場合，記号はdになります．そして，rangeを記号dの出現確率で縮小して，codeから(range×dの下限値)を引き算します．今度は0x200000の幅の中で0x1a8000に相当する記号を探すわけです．

d，c，b，bまで復号すると，rangeは0x4000になり0x10000より小さくなります．ここでrangeとcodeを256倍して，新しい符号語を1バイト読み込んでcodeに足し算します．この場合，符号語は0なのでcodeの値は増えません．あとは，同じ処理を繰り返して記号列"dcbbaaaa"を復号することができます．

レンジコーダのプログラムも簡単です．詳細はリスト1をお読みください．



適応型符号化



いままで説明した算術符号やレンジコーダは「静的符号化」といい，あらかじめ記号の出現確率を調べておいて，それに基づいて入力記号列を符号化していく方法です．静的符号化の場合，復号するときに符号化で用いた記号の出現確率が必要になります．このため，記号の出現頻度表を出力ファイルの先頭に付加する方法が一般的です．

これに対し「動的符号化」は，入力記号列の符号化を行いながら記号の出現確率を変化させる方法で,「適応型符号化」とも呼ばれています．最初はどの記号も同じ確率で出現すると仮定して，記号列を読み込みながら記号の出現確率を修正し，その時点での出現確率に基づいて記号の符号化を行います．

動的符号化の特徴は入力記号列の性質(出現確率)の変化に適応できることです．また，動的符号化では復号しながら記号の出現確率を求めることができるので，出現頻度表をファイル

〔リスト 2〕適応型レンジコーダの符号化と復号

```
#define MAX_SUM  0x8000

/* 出現頻度表の初期化 */
void init_count_table( void )
{
  int i;
  count_sum[0] = 0;
  for( i = 0; i < CODE_SIZE; i++ ){
    count[i] = 1;
    count_sum[i + 1] = count_sum[i] + 1;
  }
}

/* 出現頻度表の更新 */
void update_count_table( int c )
{
  count[c]++;
  while( ++c <= CODE_SIZE ) count_sum[c]++;
  if( count_sum[CODE_SIZE] >= MAX_SUM ){
    /* 出現頻度を半分にする */
    int i;
    count_sum[0] = 0;
    for( i = 0; i < CODE_SIZE; i++ ){
      count[i] = (count[i] << 1) | 1;  /* 0 にはならない */
      count_sum[i + 1] = count_sum[i] + count[i];
    }
  }
}

/* 符号化 */
int adaptive_range_encode( Uchar *out, Uchar *in, int size )
{
  int i, wp = 0, range = MAX_RANGE, low = 0;
  init_count_table();              /* 出現頻度表の初期化 */

  for( i = 0; i <= size; i++ ){
    int tmp = range / count_sum[CODE_SIZE];
    int c = (i < size ? *in++ : END);

    low += tmp * count_sum[c];     /* range と low の計算 */
    range = tmp * count[c];
    update_count_table( c );       /* 出現頻度表の更新 */

    if( low >= MAX_RANGE ){        /* 桁上がりのチェック */
      overflow( out, wp );
      low -= MAX_RANGE;
    }
    while( range < MIN_RANGE ){    /* range の拡張 */
      out[wp++] = low >> 16;
      low = (low << 8) & MASK;
      range <<= 8;
    }
  }
  /* 最後の出力 */
  out[wp++] = low >> 16;
  out[wp++] = (low >> 8) & 0xff;
  out[wp++] = low & 0xff;
  return wp;
}

/* 復号 */
int adaptive_range_decode( Uchar *out, Uchar *in )
{
  int wp = 0, range = MAX_RANGE, code;
  init_count_table();     /* 出現頻度表の初期化 */
  code = *in++;
  code = (code << 8) + *in++;
  code = (code << 8) + *in++;


  while( 1 ){
    int c;
    int tmp  = range / count_sum[CODE_SIZE];
    int tmp1 = code / tmp;

    /* とりあえず単純な線形探索 */
    for( c = 0; c < CODE_SIZE; c++ ){
      if( (count_sum[c] <= tmp1) && (tmp1 < count_sum[c + 1]) )
                                                                    break;
    }
    if( c == END ) break;          /* 終了 */
    out[wp++] = c;                 /* 出力 */

    code -= count_sum[c] * tmp;    /* range と code の計算 */
    range = count[c] * tmp;
    update_count_table( c );       /* 出現頻度表の更新 */

    while( range < MIN_RANGE ){    /* range の拡張 */
      range <<= 8;
      code = ((code << 8) & MASK) + *in++;
    }
  }
  return wp;
}
```

に付加する必要はありません．これは静的符号化にはない大きな利点です．

このほかにも動的符号化には有利な点があるため，ハフマン符号を動的符号化に対応させた適応型ハフマン符号が考案されています．しかしながら，適応型ハフマン符号は実装方法が難しく，処理速度も遅いという欠点があります．これに対し，適応型レンジコーダは簡単な方法で実装することができ，処理速度も遅くありません．

適応型レンジコーダのプログラムをリスト 2に示します．本稿のプログラム（bsrc）は適応型レンジコーダを使用しています．そして，ブロックソート向きの「情報源モデル」を作成することで圧縮率を改善しています．



情報源モデルを改良する

圧縮アルゴリズムは「モデル化」と「符号化」という二つの部分に分けて考えることができます．モデルは入力された記号列から作成され，各記号の出現確率を求めます．たとえば，記号aの確率は1/10で，記号bの確率は9/10のように決定します．符号化は確率に基づいて符号語を割り当て，入力された記号を符号化して出力します．

モデル化にはいろいろな方法があり，ブロックソートもその一つです．適応型レンジコーダの場合，記号を読み込みながらモデルを更新し，その時点での確率に基づいて符号化を行っています．モデルを更新しながら符号化を行うのところが適応型符号化の特徴です．適応型レンジコーダは単に各記号の出現確率を求めているだけの簡単なものですが，これもモデル化の一つです．

このような簡単なモデルを「無記憶情報源モデル」といいます．情報源は記号を生成する元（発生源）と考えてください．情報源が記号を生成するとき，以前に生成した記号との間に関係がないことを「無記憶」といいます．簡単にいえば，記号tの次はhが出るとか，t，hと続いたら次はeが出るといった関係はなく，確率でのみ記号が生成されるということです．静的なレ

ンジコーダ(算術符号)やハフマン符号も無記憶情報源モデルになります.

ブロックソートの場合，MTF法で変換することにより記号0がとても多くなるため，無記憶情報源モデルの適応型レンジコーダでも効率よく圧縮することができます．ですが，ブロックソートとMTF法の特徴はそれだけではありません．1や2などの小さな記号も多くなり，逆に0xfeや0xffなどの大きな記号はとても少なくなります．つまり，記号が大きくなるにしたがって個数が減少していく反比例の関係になります.

このあとZero Length Encodingを適用すると0と1の個数は減少しますが，それでも記号0の個数は多くて，反比例の関係に大きな変化はないと考えられます．もしも，このような関係に適した情報源モデルを作成できれば，ブロックソートの圧縮率を改善することができるでしょう.

そこで，ブロックソートとMTF法で変換されたデータに適したモデルを作成することにします．本稿のプログラム(bsrc)では圧縮ツールZzip[8]のモデルを参考にしました．最初のポイントは，表3のように記号をグループに分けるところです．Group 0, 1, 2は一つの記号に対応しますが，Group 3は3と4で，Group 4は5, 6, 7, 8というように，グループに割り当てる記号の個数を増やしていきます.

次に，記号をGroup番号(first code)とGroup内の番号(second code)の二つに分けて符号化します．つまり，first code(0～9)をレンジコーダで符号化し，次にsecond codeを符号化するのです．これが第2のポイントです．そして最後のポイントが，second codeを符号化するとき，グループによって記号の出現頻度表を切り替えるところです.

Group 0, 1, 2は一つの記号を表すのでsecond codeは必要ありません．その分だけ記号0, 1, 2は効率よく符号化することができます．そして，first codeは0から9までの10種類しかありません．記号の全種類を符号化するよりも，圧縮率は大幅に向上するはずです．Group 3から9はsecond codeが必要になりますが，その増加分を上回るほどfirst codeの圧縮率が良ければ，結果として圧縮率は向上するというわけです.

実際にプログラムを作って試してみたところ，圧縮率は確かに向上するのですが，その効果はわずかなものでした．そこで，first codeに「有限文脈モデル」を適用することにします.

有限文脈モデル

「無記憶情報源モデル」はもっとも簡単な情報源モデルですが，このモデルを一般化して状態(記憶)を持つモデルを考えることができます．このようなモデルを「有限状態確率モデル」とか「マルコフ情報源モデル」と呼びます.

簡単に説明すると，情報源にはいくつかの状態があって，その状態によって記号の生成確率が異なります．そして，ある記号が生成されると別の状態へ移動します．これを「状態遷移」といいます．このようなモデルは状態遷移図で表すことができます．簡単な例を図5に示します.

〔表3〕記号のグループ分け

GR	Num	Code
0	1	0
1	1	1
2	1	2
3	2	3, 4
4	4	5, 6, 7, 8
5	8	9～16
6	16	17～32
7	32	33～64
8	64	65～128
9	128	129～256(256: 終端記号)

〔図5〕簡単な状態遷移図

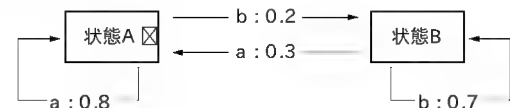


図5では，記号がaとbの2種類あり，二つの状態AとBがあります．AとBでは記号の出現確率が異なることに注意してください．そして，Aの状態で記号bが出力されると，状態はBへ移ります．記号aが出力されても状態はAのままです．逆に，状態Bで記号aが出力されると，状態はAに移ります．記号Bが出力されても状態は移りません.

このモデルの場合，AとBともに状態遷移する確率が低いので，aaaaaaabbbbbbbのように同じ記号が連続して出力される確率がとても高くなります．そして，この記号列を無記憶情報源モデルで符号化しても，効率よく圧縮できないことはすぐにわかると思います.

このような場合，状態によって記号の出現頻度表を切り替えることで，効率よく圧縮することができます．つまり，状態Aの出現頻度表Table Aと状態Bの出現頻度表Table Bを用意し，状態AではTable Aを，状態BではTable Bを使って符号化すればいいわけです.

このように，モデルが決まっていれば簡単なのですが，一般的なデータで有効なモデルを作成することはとても難しいことです．そこで，次のような単純なモデルを考えます.

「生成される記号の確率は直前に出現した記号列によって定められる」

これを「有限文脈モデル」といいます．そして，直前に出現した記号列の長さを次数(order)といいます.

有限文脈モデルは1次(order-1)がいちばん簡単です．直前に出力した記号を覚えておいて，それに従って出現頻度表を切り替えるという単純な方法で実現できます．つまり，各記号ごとに出現頻度表を用意しておいて，直前に出力した記号がaであれば，aの出現頻度表を使って符号化を行います.

〔表4〕The Canterbury Corpus の評価結果

ファイル名	サイズ	LHA	bzip2	bsrc	szip	zzip
alice29.txt	152,089	59,117 (38.9)	43,202 (28.4)	43,230 (28.4)	42,835 (28.2)	41,962 (27.6)
asyoulik.txt	125,179	52,341 (41.8)	39,569 (31.6)	39,919 (31.9)	39,431 (31.5)	38,747 (31.0)
cp.html	24,603	8,384 (34.1)	7,624 (31.0)	7,625 (31.0)	7,541 (30.7)	7,509 (30.5)
fields.c	11,150	3,170 (28.4)	3,039 (27.3)	3,023 (27.1)	3,086 (27.7)	2,971 (26.6)
grammar.lsp	3,721	1,271 (34.2)	1,283 (34.5)	1,244 (33.4)	1,232 (33.1)	1,243 (33.4)
kennedy.xls	1,029,744	198,342 (19.3)	130,280 (12.7)	99,949 (9.7)	107,591 (10.4)	24,778 (2.4)
lcet10.txt	426,754	159,558 (37.4)	107,706 (25.2)	108,013 (25.3)	106,884 (25.0)	104,193 (24.4)
plrabn12.txt	481,861	210,045 (43.6)	145,577 (30.2)	146,601 (30.4)	143,631 (29.8)	141,418 (29.3)
ptt5	513,216	52,305 (10.2)	49,759 (9.7)	48,794 (9.5)	52,858 (10.3)	48,329 (9.4)
sum	38,240	13,993 (36.6)	12,909 (33.8)	12,599 (32.9)	12,851 (33.6)	12,297 (32.2)
xargs.1	4,227	1,778 (42.1)	1,762 (41.7)	1,711 (40.5)	1,739 (41.1)	1,710 (40.5)
合計	2,810,784	760,304 (27.0)	542,710 (19.3)	512,708 (18.2)	519,679 (18.5)	425,157 (15.1)

圧縮ツールで使用したオプション　　単位：バイト，( )は圧縮率[%]

lha a
bzip2 -b9
bsrc -e
szip -b17
zzip a -1m

〔表5〕The Large Corpus の評価結果(バッファサイズ1Mバイト)

ファイル名	サイズ	bzip2	bsrc	szip	zzip
bible.txt	4,047,392	845,623 (20.9)	841,671 (20.8)	840,582 (20.8)	811,239 (20.0)
Canterbury	2,821,120	568,468 (20.2)	523,347 (18.6)	530,564 (18.8)	539,553 (19.1)
e.coli	4,638,690	1,251,004 (27.0)	1,233,563 (26.6)	1,174,473 (25.3)	1,167,753 (25.2)
world192.txt	2,473,400	489,583 (19.8)	482,329 (19.5)	508,380 (20.6)	469,101 (19.0)
合計	13,980,602	3,154,678 (22.6)	3,080,910 (22.0)	3,053,999 (21.9)	2,987,646 (21.4)

圧縮ツールで使用したオプション　　単位：バイト，( )は圧縮率[%]

bzip2 -b9
bsrc -e
szip -b11
zzip a -1m

〔表6〕The Large Corpus の評価結果(バッファサイズ2Mバイト)

ファイル名	サイズ	bsrc	szip	zzip
bible.txt	4,047,392	818,883 (20.2)	813,011 (20.1)	787,428 (19.5)
Canterbury	2,821,120	536,825 (19.0)	532,069 (18.9)	542,594 (19.2)
e.coli	4,638,690	1,230,668 (26.5)	1,174,122 (25.3)	1,166,478 (25.1)
world192.txt	2,473,400	448,518 (18.1)	483,457 (19.5)	435,393 (17.6)
合計	13,980,602	3,034,894 (21.7)	3,002,659 (21.5)	2,931,893 (21.0)

圧縮ツールで使用したオプション　　単位：バイト，( )は圧縮率[%]

bsrc -e -b4
szip -b21
zzip a -2m

したがって，記号が255種類あれば出現頻度表も255個必要になります．order-2であれば，abやcdのあとに現れる記号の出現頻度表が必要になるので，個数は65536になります．このように，次数が大きくなるほど必要となるメモリ量が爆発的に増えるので，単純な方法では低次の有限文脈モデルしか実現できないのが欠点です．

ところが，今回のモデルでは記号を10 Groupに分けるので，first codeは10種類しかありません．したがって，高次の有限文脈モデルでも簡単に試してみることができます．実際に試してみると，order-2でbzip2に匹敵する圧縮率を達成することができました．詳細はソースファイル(bsrc.c)をお読みください[注1]．



評価結果は？

それではプログラム(bsrc)の評価結果を示します．圧縮率を比較するために，LHA，bzip2，szip[9]，zzipの評価結果も示します．LHA以外の圧縮ツールはすべてブロックソートを使っています．テストデータはCanterbury Corpus[10]で配布されているThe Canterbury CorpusとThe Large Corpusを使いました．

The Canterbury Corpusの評価結果を表4に示します．ブロックソートを使った圧縮ツールはどれもLHAより圧縮率が高く，ブロックソートがとても優れた圧縮アルゴリズムであることがわかります．bsrcはおおむねbzip2と同等の圧縮率を達成していますが，`kennedy.xls`の圧縮率が高い分だけ合計の圧縮率はbzip2とszipよりもよくなりました．zzipは`kennedy.xls`に対して特別な操作を行っているため，極端に圧縮率がよくなっています．zzipはソースが公開されているので，興味のある方はソースを読んでみてください．

次はThe Large Corpusの評価結果を示します．バッファの大きさを1Mバイトに設定した結果を表5に，2Mバイトに設定した結果を表6に示します．CanterburyはThe Canterbury Corpusのファイルをアーカイバtarでまとめたものです．bzip2のバッファは最大で900Kバイトまでしか設定できないので，

注1：`bsrc.c`およびコンパイル済みの実行プログラム，今回掲載したソースリストは`http://www.cqpub.co.jp/interface/`よりダウンロードできるほか，本誌付属のCD-ROMへも収録されている．

今回のテストは少々不利であることに注意してください．

bsrcの圧縮率はbzip2よりも高くなりましたが，バッファサイズの差が影響しているかもしれません．合計ではszipやzzipにはかないませんが，`e.coli`以外のファイルは高い圧縮率を達成しています．bsrcではMTF法，ランレングス，情報源モデルを改良しましたが，その効果は十分に出ていると思います．

ブロックソートの場合，バッファサイズを大きくした方が圧縮率は良くなると考えられますが，結果を見ると例外もあることがわかりました．バッファを2Mバイトに増やした場合，どのツールでもCanterburyの圧縮率は低下しています．じつは，The Canterbury Corpusの`kennedy.xls`をbsrcで圧縮する場合，バッファサイズを512Kバイトにした方が圧縮率は高くなります．

ブロックソートの変換結果は，データの区切り方によって異なります．データによっては，それが圧縮率に大きな影響を与える場合もあると考えられますが，詳しいことは筆者もよくわかりません．今後の課題にしたいと思います．

おわりに

今回はブロックソートとレンジコーダを使ってファイルの圧縮プログラム(bsrc)を作成しました．ブロックソートしたあと，MTF法とランレングスでデータを変換し，レンジコーダで符号化するという一般的な方法でファイルを圧縮しましたが，MTF法，ランレングス，情報源モデルを改良することで，bzip2に匹敵する圧縮率を達成することができました．特別な方法ではなく，基本的なアルゴリズムの組み合わせだけでここまで圧縮できるとは筆者も大変驚いています．

ところで，本稿で試した方法がベストというわけではありません．とくに情報源モデルの改良では，もっと優れたモデル化があると思います．また，このほかにも圧縮率を高める方法があるでしょう．幸いにして，ソースを公開している圧縮ツールが多数(bzip2，bwtzip，szip，zzipなど)あります．興味のある方はぜひソースを読んでみてください．

最後に，本稿がデータ圧縮アルゴリズムに関心をもたれている読者の参考になれば幸いです．

● 権利・免責事項など

本稿で作成したプログラムbsrcはフリーソフトウェアとします．ご自由にお使いください．ただし，これらのプログラムは無保証であり，使用したことにより生じた損害について，筆者は一切の責任を負いません．また，これらのプログラムを販売することで利益を得るといった商行為は禁止いたします．

bsrcはデータ圧縮アルゴリズム評価用のサンプルプログラムであり，ファイルの安全性はまったく考慮しておりません．実用的な圧縮ツールとして使用しないようお願いいたします．

参考文献とURL

1) 植松友彦,『文書データ圧縮アルゴリズム入門』，CQ出版(株)，1994
2) 奥村晴彦,『C言語による最新アルゴリズム事典』，技術評論社，1991
3) 奥村晴彦,「データ圧縮の基礎から応用まで」,『C MAGAZINE』，2002年7月号
4) The bzip2 and libbzip2 home page，http://sources.redhat.com/bzip2/
5) 大規模テキスト索引(suffix array)の構築法とその情報検索への応用 suffix array構築アルゴリズムと実装，http://www.gi.k.u-tokyo.ac.jp/ssr-homepage/1999/workshop1/sadakane/
6) The iss Homepage，http://www-imai.is.s.u-tokyo.ac.jp/~sada/iss/
7) bwtzip，http://stl.caltech.edu/bwtzip.shtml
8) Zzip，http://debin.org/zzip/
9) szip homepage，http://www.compressconsult.com/szip/
10) Canterbury Corpus，http://corpus.canterbury.ac.nz/
11) M.Hiroi's Home Page，http://www.geocities.co.jp/SiliconValley-Oakland/1680/

ひろい・まこと

小型・軽量でSDメモリカードと互換性もある

SDIOカード開発入門

第3回 802.11b 無線LANとSDIOカード

井出 裕

はじめに

連載第3回目となる今回は，IEEE規格および一般的な無線LAN技術の解説を行いながら，SDIO無線LANカード SD-Link11b（写真1，シイガイズ）を例に取り，SDIO無線LANカードの概略について解説する．これからSDIOカードの開発を検討している読者の参考となれば幸いである．

無線LANとSDIOの通信速度

無線機器の性能を示すものの一つとしてデータ伝送速度が挙げられる．伝送速度は，電気信号が変化する速度（変調速度と呼ばれ，1秒間に何回変調をかけられるかの速度．単位はbps）と，1回あたりの変化で表現できる情報量（ビット数）で決まり，1秒間にどれほどの情報量を送ることができるかを示している速度である．

ディジタル信号を送るのに必要な帯域は，1秒間に送るパルスの数で決まる．理論的には帯域の2倍のパルスまで送ることが可能だが，実用的な条件下では帯域と同じ程度の数しか送れない．

また，SDIO標準規格Ver1.0ではデータ伝送方式として3通りの方法が定義されている．一つ目はSPIモードで，これはIC間通信で広く採用されているシリアル通信である．二つ目はSD1ビットモードでシリアルでの通信，三つ目のSD4ビットモードではSD1ビットモードを四つ並列に使用するイメージとなる．

1）低速度でのSDIOへの応用

SDIOでは1ビットか4ビットでのデータ転送が決められている．その場合の最大クロック周波数は25MHzであり，バスのサイズとクロック周波数で転送速度が決まる．もし1ビットのバスと20MHzのクロックを利用した場合，そのデータ転送速度は20Mbpsになる．

2）高速度でのSDIOへの応用

もし4ビットバスと25MHzのクロックが使用された場合は，100Mbpsの転送速度になり，SDIOで規定されているもっとも速い速度となる．

いうまでもないが，低速のSDIO通信を選択した場合，高速のSDIO通信（100Mbps）を必要とする周辺機器間とのデータ転送はできなくなる．IEEE802.11bでは11Mbpsの最高速度が規定されており，SDIO転送でも通信が可能となる．速度は通信機器では大事な要因となり，製造会社はそれぞれの用途に合ったモードを選択し，規格で定められている最大速度を十分に活用して通信できる製品を開発することが望ましい．

無線LANをSDIOカードで実現する

PDAやPCなどのSDホストのSDIOインターフェースに使用されるNICは，I/Oシグナルをあるバスから SDバス（またはその逆）へと変換させる機能をもっている必要がある（図1）．

SD-Link11bは，SDIOスロットに差し込むだけで内蔵アンテナを使った無線LAN通信が可能になり，NICとして使用できる製品である．このカードの内部ブロック図を図2に示す．SDIO無線LANは，その無線機能とSDホスト間をSDバスにより接続するが，そのためにはSDバスと接続できるインターフェース回路が必要である．そこにはSDIOコントローラデバイス SD-Path CG100（写真2，シイガイズ）が使われている．

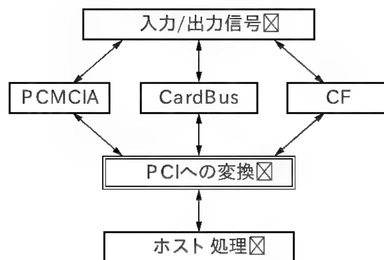
CG100を使用した場合，無線機能とのインターフェースなどで使用されているPCMCIAなどの信号からCG100により，SDバスに変換（またはその逆）されるため，外からはあたかもこれらが直結されているように見える．

SDIOカードの場合，先に触れたSDIOインターフェース回

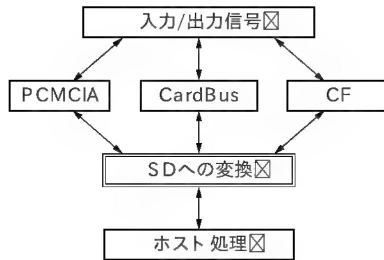
〔写真1〕SDIO無線LANカード SD-Link11b（シイガイズ）



〔図1〕NICの信号変換について

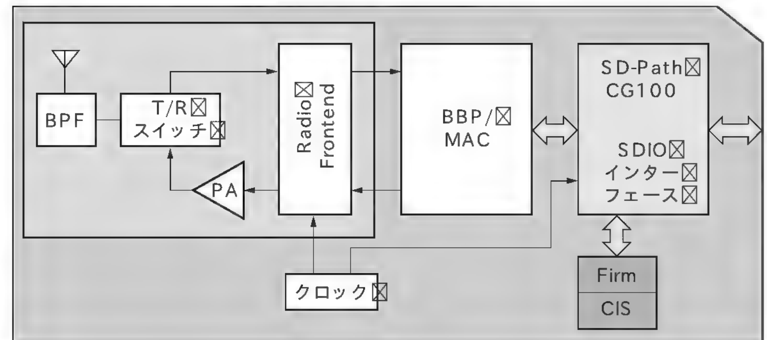


(a) 標準的な無線LANのNIC



(b) 標準的なSDIOのNIC

〔図2〕SD-Link11bカードブロック図



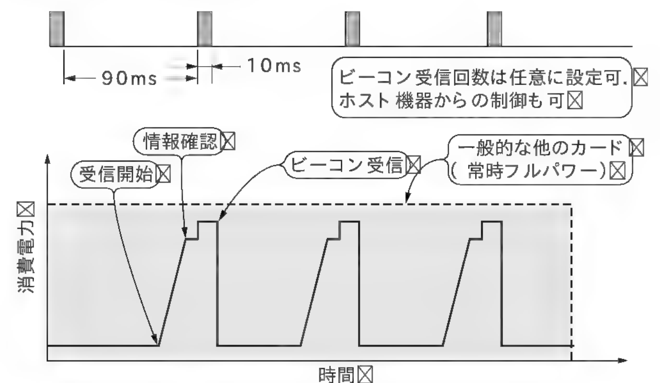
〔写真2〕SDIOコントローラデバイスCG100



路の設計も重要なポイントとなるが，SDIOカードはバッテリ駆動機器で使用される例が多く見込まれるため，もう一つの重要なポイントとして，特に消費電力について考慮する必要がある．

SD-Link11bでは動作モードとして通常通信モード，待ち受けモード，そしてディープスリープモードの3種類を用意した(図3)．待ち受けモードは図のようにビーコン受信タイミングにあわせたRX受信部の動作を行うことにより，受信時以外の消費電力を13mA以下に抑えており，さらにディープスリープモードでは1秒前後での再起動性能をもちながら，消費電力を2mA以下に抑えている．Webのブラウズなどを想像してもらえればわかると思うが，実際の通信時間は画面コンテンツを読んでいる(または見ている)時間と比較して遥かに少ないことが理解してもらえると思う．SD-Link11bはこれらのアイドル時間の消費電力を削減する機能をもたせたSDIO無線LANカードである．

〔図3〕SD-Link11bの動作モード



SDIO無線LANカードの開発

SD-Link11bは内蔵方法を簡単にするため，現存の無線LANのチップをカードの中に組み込んでいる．IEEE802.11bのMAC機能を操作するためのドライバはSDホストに収納され，SDホストで使用されているOS(Windowsであれば NDIS)により制御される．SDホストからは，あたかもSDIOのレジスタ空間に無線LANのMAC部のレジスタが割り当てられ，それらが直接制御できるように見える．

ここでポイントとなるSD信号と無線LAN信号の変換では，次のような内容を満足させる必要がある．

1) 高効率のSDIOコントローラデバイスを使用すること
2) 消費電力が少ないこと
3) 信号のバス変換をする際に余計な労力を使わないこと
4) インターフェース部分の開発は一般性を持たせ，どのようなIEEE802.11bのデバイスでも対応可能であること

これら注意事項を盛り込んだ場合のSDIO無線LANカードの代表的なブロックを図4に示す．

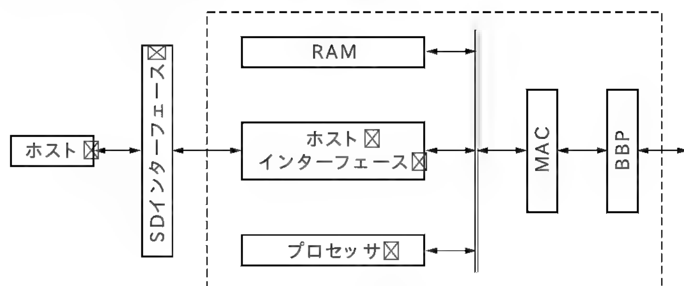
MACのデータフレームの処理

SDIOによる無線LANの制御では，SDバスを使用して間接的にMAC部分を操作するわけだが，MACの中をどのように

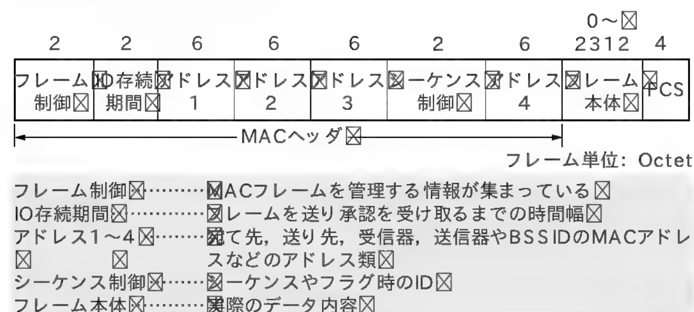
制御するかについては，MACの中の動作を見る必要がある．IEEE802.11bで規定されているMAC層のデータは次のような内容から作成されている（図5）．

1）MACのヘッダにはフレーム管理，時間，アドレスとシーケンス管理についての情報が含まれる
2）フレームの種類によりフレームの長さが変化する
3）32ビットのCRCを含むフレームCheckシーケンス

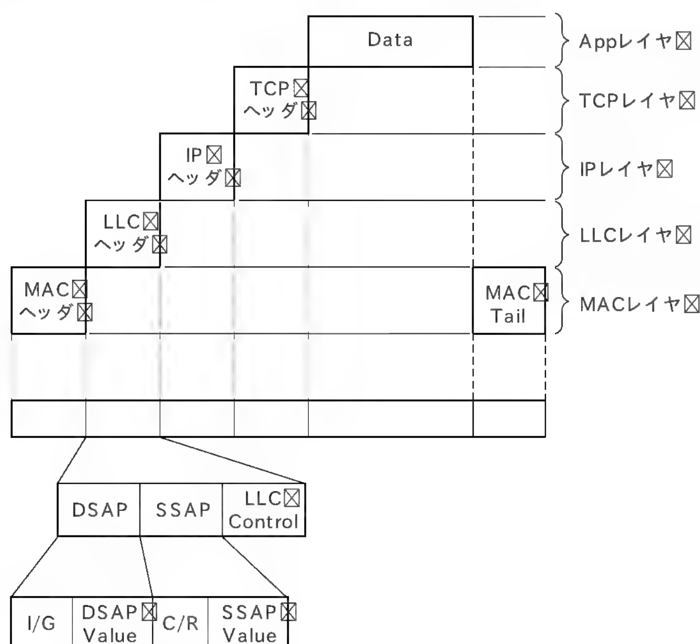
〔図4〕SDとのインターフェース機能ブロック



〔図5〕MACフレームの内容



〔図6〕LLCヘッダの構造



MAC内で使用されるフレームはMSDU（MAC Service Data Unit）またはMMPDU（Multi MAC Protocol Data Unit）と呼ばれ，これらのフレームは一般的に長くなる傾向が強い．そこで転送しやすいようにフレームを短くしたフレームがある．これはMPDU（MAC Protocol Data Unit）と呼ばれ，送信が簡単になるだけではなく，短くすることによりデータ内容がそのままの形で送り届けられ，信頼性も上がるという利点がある．しかし短いフレームが多すぎると，各々のフレームの上にヘッダが添付されるため，その部分のオーバヘッド処理も増えることになる．SDIOにおいて最適な状態とは，これらヘッダ処理の釣り合いの取れた処理環境を作り上げることによって，間接的に消費電力とデータ伝送速度の改善を実現することである．

MACフレームの中にある物理層収束プロトコル（PLCP）とロジカルリンク層制御（LLC）がSD-Link11bの主役となる．MSDUがLLCに送られる（または受信）の場合，長いMSDUのパケットはこの部分で短くされる．したがってSDIOのコマンドでこの部分の操作ができることが必要となる（図6）．

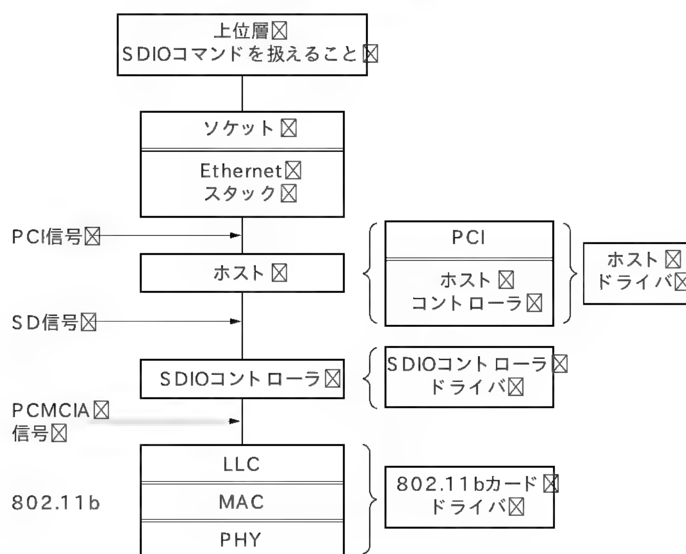
DSAP（Destination Access Point）は宛て先SAP，SSAP（Source Service Access Point）は発信元SAP，I/G（Indivisual/Group）は個別/グループアドレスの区別，C/R（Common/Response）は共通/応答アドレスの区別を意味する．実際は，LLCのTCPに関する部分についてはNDISなどOSに依存する部分があり，その部分に手を付ける必要はないが，IPに関する部分は機器に依存しており，この部分のSDIOによる操作が必要となる．

無線LANデータ処理の流れ

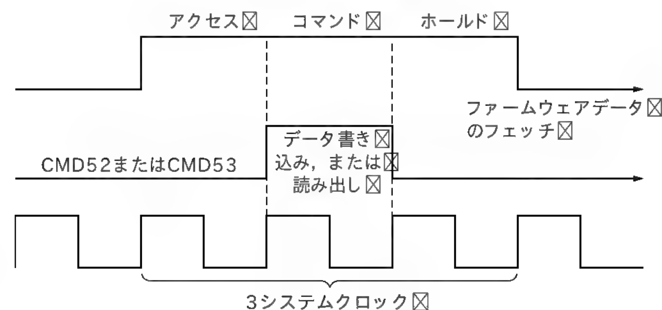
無線LANは"LAN"ということもあり，基本的にEthernetなどの一般的なネットワーク信号処理構造と大きな違いはない．しかし，リンク層（LLC）以降のバス信号接続が，PCなどであればPCIなどに直接接続されるところが，SD-Link11bではこの部分がPCMCIAを経由し，SDバスで接続され，SDホストを通じて最終的にPCIへ接続されている（図7）．もちろんSDホスト以降は，そのSDホスト機器によりPCIやローカルバスなどそれぞれ異なるが，ここではPCIへ接続される一般的なPC（パソコン）を想定して説明する．

さて，無線LANがデータを受信すると，物理層（PHY）を経由して無線信号をデータ化した後に，先に説明したMACフレームがMAC部に保持され，MAC部より受信データがある旨を伝えるための割り込み信号が発生する．この割り込み信号は，MACとSDIOコントローラ間のPCMCIAバスを通じてSDIOコントローラに伝達され，さらにSDIOコントローラからSDバスを通してSDホストコントローラに伝達される．SDホストで割り込みを検知すると，PCIバスを通してCPUに割り込みが伝達され，OSが割り込みを検知すると，SDホスト側に搭載されているドライバの割り込み処理がスタートする．

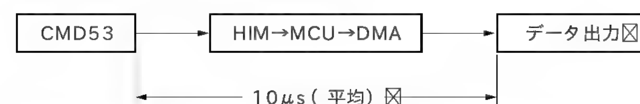
〔図7〕SDホストとカード間での信号変換



〔図8〕コマンドとシステムクロックの関係



〔図9〕SDIOコントローラ内の動作



・HIM（Host Interface Module)は，SDバスからコマンドが入力されると，それに対するレスポンスを返すと同時にMCUに割り込みを出す
・MCU（Multipoint Control Unit)は8ビットのコントローラ（マイコン）
・DMA（Direct Memory Access)は，リード/ライト用FIFOと，PCカードインターフェース間のデータ転送を行う

またSDホストからSDIOカードへの各種信号は，逆の経路を取りSDIOカードをSDホスト機器のアプリケーションソフト上で動作させることが可能だが，SDホスト機器とカードという異なる2種類のハードウェア同士でのデータのやり取りは，SDIOで規定されているコマンドを介して初めて可能になるということを忘れてはならない．

ここで，SDIOコントローラとMAC部の間は，PCMCIAインターフェースで接続されている．このインターフェースは16ビット幅だが，16ビットのデータを転送するのに最低3システムクロックを要する（図8）．したがって，一般にここがネックにならないように設計する．

100Mbpsにできるだけ近い速度が得られるようなシステムに設計する必要がある．ただし，常にPCMCIAインターフェースのバスが理想的に動作しているとは限らないので，さまざまな遅延要因も考慮した速度に設定すれば実用的にも問題はないだろう．

CG100を例にした具体的なSDIOコントローラの動きは，次のようになる（図9）．

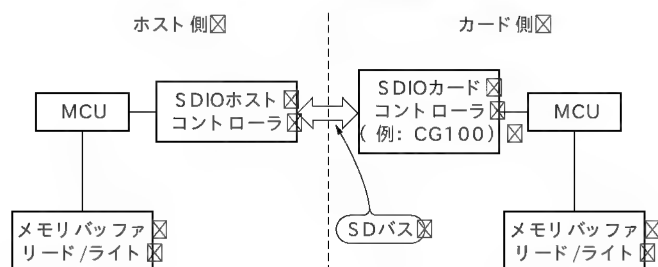
1) SDホストよりコマンドが送られる．コマンドの内容はそのタイプ，情報，送信されるデータ事項の数と宛て先が含まれる
2) HIM（Host Interface Module)は，SDバスからコマンドが入力されると，それに対するレスポンスを返すと同時にCG100内蔵MCUに割り込みを出す
3) ライトコマンドについてはSDホストがレスポンスを受け取った時点でデータが流れ出す
4) HIMが一度メモリバッファ（内蔵FIFO)へライトデータを収納する．その間割り込みがかかったCG100内蔵MCUはDMAを操作し，インターフェースの用意をする
5) ライトデータがすべてSDIOコントローラに送られると，HIMはSDホストにデータレスポンスを返しCG100内蔵MCUにはライトデータが揃ったことを示す割り込みを発生する
6)「データ受信OK」のメッセージをSDホストに出し，CG100内蔵MCUはDMAを開始する

ここでFIFOとは，あるシステムから異なるシステムへデータを転送する際のバッファに使用され，複数のソースから共有バスへデータを流すときに一時的にそのデータを収納する役目を担う．バスが既に使用されているときはデータがFIFOに収納され，バスが空いているときはそのデータをバスへ流すようになっている．

SDIO無線LANカード開発の難点と問題点

無線LANにおける無線性能はもちろん無線部（RF)の性能に依存するわけだが，汎用チップセットを使用する以上，無線部に関してはSDIOカードと他カードの違いは実装による特性の問題を除けばそれほどはない．問題は信号の流れであり，無線LANチップセット→PCMCIA→SDIOコントローラデバイス→SDバス→SDホストデバイス→PCI→MCUのように多くのデータ変換工程を経てホスト側MCUに到達しなければならないことである．さらに一般のPCカードには直接的にアドレスとデータおよびリード/ライト信号でアクセスできるのに対し，

〔図10〕バッファ活用によるアイドル時間の削減



SDIOカードではSDコマンドに対するレスポンスの返信やSDコマンド/レスポンスに引き数としてアドレスやデータおよびリード/ライトなどの指示子を入れなければならず，間接的なやり取りを強いられる構造になっている．この一連の流れの中で，いかに効率よく各ファンクション間でデータ転送を行えるかで性能が決定するといっても過言ではない．

さて，ここで有力な解決策としてはメモリバッファとしてのFIFOの活用である．これらは，データ変換を比較的多く行うシステムや，相手が無線のような場合において受信側が常にFIFOという理想的な受信体制をもつことにより，パケットロスや再送信などのむだを省くことに大きく寄与している．したがってFIFOの使用は限定領域のネットワーク，無線通信や狭範囲での通信に適しており，とくにSDIOでの無線LANシステムには最適といえる．以下にこれらの問題点を解決するためのポイントをまとめた．

1) SDIOのコントローラは十分なデータ処理能力と効率性が必要
2) SDIOのSDホストもデータ処理能力と効率性が必要
3) 信号のやり取りに応じた十分な性能と規模をもつFIFOを使用すること

具体的には，これらは内部で使用されているMCU，バッファの大きさやデータをどのように処理するかに依存しており，アイドル時間が長くなっている部分はシステム上のボトルネックを意味する．さらにシステムとして見た場合，SDホスト側の信号処理能力も一つのボトルネックになり得る．そのほか，SDホスト側とカード側のクロックの整合性，電源供給，イニシャライゼーションの方法など詳細部分についての検討が必要とされる．

理想的なシステムはこれらの点が改善されており，複数のバスを休ませることなく常に作動させ，しかも電力を極力抑えるようなバッファがSDホスト側とカード側にあれば，常にこれらのバッファを作動させて効率の良いシステムができる(図10)．

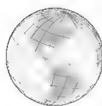
まとめ

ここまで解説したように，IEEE802.11bのような広帯域を扱うカードにおいては最適なSDIOコントローラを選択することとソフトウェアを最適化することが重要となる．広帯域を要求されるアプリケーションは無線LANだけではなく，高画素のディジタルカメラや地上ディジタルTV放送チューナSDIOカードなど多岐におよぶ．

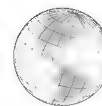
＊　　　＊

今回はIEEE802.11b無線LANの基本にふれながら，SDIO無線LANカードSD-Link11bを例に挙げ，SDIO無線LANカード開発におけるポイントなどについて解説を行った．次回からは，SDIOを開発するために重要な要素であるSDIOコントローラデバイスについて，実際に動作させるまでの解説を2回に分けて行う．

いで・ひろし　シイガイズ(株)プロダクトR&Dディレクタ



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

12/1-3 **2003 Taipei Power Forum & Exhibition**
Taipei International Convention Center, Taipei, Taiwan
Materials Research Laboratories, ITRI
http://www.mrl.itri.org.tw/battery/default_e.htm

12/7-10 **2003 IEEE International Electron Devices Meeting**
Hilton Washington and Tower, Washington, D.C., WA, USA
Institute of Electrical and Electronics Engineers, Inc.
http://www.his.com/~iedm/

12/9-11 **Bluetooth Americas 2003**
San Jose Convention Center, San Jose, CA, USA
Informa UK Limited
http://www.ibctelecoms.com/bluetoothamericas/

12/9-12 **42th IEEE Conference on Decision and Control**
Hyatt Regency Maui, Maui, HW, USA
Institute of Electrical and Electronics Engineers, Inc.
http://www2.acae.cuhk.edu.hk/~ycliu/cdc03/

12/15-19 **5th Pacific Rim Conference on Lasers and Electro-Optics**
Taipei International Convention Center, Taipei, Taiwan
CLEO/PR 2003
http://cc.ee.ntu.edu.tw/~cpr2003/

2004 1/6-8 **Electronics West Show**
Anaheim Convention Center, Anaheim, CA, USA
Canon Communications LLC
http://www.devicelink.com/expo/ewest04/

1/8-11 **2004 International CES**
Las Vegas Convention Center, Alexis Park, Las Vegas Hilton, Las Vegas, NV, USA
CES Customer Service
http://www.cesweb.org/

国内イベント

12/2-5 Internet Week 2003
パシフィコ横浜(神奈川県横浜市)
Internet Week 2003 登録事務局
http://internetweek.jp/

12/3-5 '03 国際画像機器展
パシフィコ横浜(神奈川県横浜市)
精機通信社
http://www.seiki-tsushin.com/ite/

12/11-13 TRONSHOW 2004
東京国際フォーラム(東京都千代田区)
トロンシンポジウム実行委員会事務局
http://www.tron.org/show.html

12/12 第6回 五大都市FPGAカンファレンス2003
一博多FPGAカンファレンス
アクロス福岡(福岡県福岡市)
セイコーインスツルメンツ内FPGAコンソーシアム事務局
http://www.sii.co.jp/eda/fpga/

12/16-17 2003年映像情報メディア学会冬季大会
工学院大学 新宿キャンパス(東京都新宿区)
映像情報メディア学会事務局 冬季大会係
http://www.ite.or.jp/taikai.html

12/19-21 第4回 計測自動制御学会システムインテグレーション部門講演会
東海大学 代々木キャンパス(東京都渋谷区)
計測自動制御学会事務局 部門協議会担当
http://srv02.sice.or.jp/~si-div/si2003/

2004 1/15-16 2004年情報学シンポジウム「ユニバーサルとユービキタス」
日本学術会議講堂(東京都港区)
情報処理学会 シンポジウム係
http://www.ipsj.or.jp/katsudou/sig/sighp/fi/

セミナー情報

リアルタイム制御のためのDSP活用テクニック
開催日時 : 12月4日(木)~5日(金)
開催場所 : オームビル(東京都千代田区)
受講料 : 68,800円
問い合わせ先:(株)トリケップス, ☎(03)3294-2547, FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/030929a.htm

CMM(CMM Integrated)の導入法と実践法[演習つき]
開催日時 : 12月9日(火)~10日(水)
開催場所 : SRCセミナールーム(東京都高田馬場)
受講料 : 76,000円
問い合わせ先:(株)ソフト・リサーチ・センター, ☎(03)5272-6071
http://www.src-j.com/teiki_no/Src/cmmi_2.htm

New!!ASPによるWebアプリケーション開発[基礎から実践まで]
開催日時 : 12月10日(水)~12日(金)
開催場所 : SRCセミナールーム(東京都高田馬場)
受講料 : 114,000円
問い合わせ先:(株)ソフト・リサーチ・センター, ☎(03)5272-6071
http://www.src-j.com/training_no/asp_web_1.htm

Verilog-HDL入門
開催日時 : 12月11日(木)
開催場所 : CQ出版セミナールーム
受講料 : 13,000円
問い合わせ先: エレクトロニクス・セミナー事務局, ☎(03)5395-2125, FAX(03)5395-1255

VC++&Win32APIによるWindowsプログラミング基礎-1
開催日時 : 12月11日(木)~12日(金)
開催場所 : ディーアイエステクノサービス研修室(東京都文京区)
受講料 : 94,000円
問い合わせ先:(株)エイチアイ ICP事業部, ☎(03)3719-8155, FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/c-vc/vc_win1.asp

Linuxデバイスドライバ開発入門
開催日時 : 12月12日(金)
開催場所 : CQ出版セミナールーム
受講料 : 13,000円
問い合わせ先: エレクトロニクス・セミナー事務局, ☎(03)5395-2125, FAX(03)5395-1255

JavaSwingプログラミング入門
開催日時 : 12月15日(月)
開催場所 : ディーアイエステクノサービス研修室(東京都文京区)
受講料 : 42,000円
問い合わせ先:(株)エイチアイ ICP事業部, ☎(03)3719-8155, FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/Java/j_swing.asp

LPI認定(LPIC)Level1 Release2対応Linux入門 入門編
開催日時 : 12月17日(水)
開催場所 : ディーアイエステクノサービス研修室(東京都文京区)
受講料 : 26,250円
問い合わせ先:(株)エイチアイ ICP事業部, ☎(03)3719-8155, FAX(03)3793-5109
http://icp.hicorp.co.jp/seminar/lpi/lpi_intro.asp

移動体通信と利用技術
開催日時 : 12月17日(水)~18日(木)
開催場所 : 高度ポリテクセンター(千葉県千葉市)
受講料 : 35,000円
問い合わせ先: 雇用・能力開発機構 高度ポリテクセンター事業課, ☎(043)296-2582
http://www.apc.ehdo.go.jp/

若手エンジニアに向けた「回路シミュレーション技術とMOSFETモデリング」
開催日時 : 12月17日(水)~18日(木)
開催場所 : ホテル機山館(東京都文京区)
受講料 : 80,750円
問い合わせ先:(株)トリケップス, ☎(03)3294-2547, FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/s031217b.htm

やり直しのためのディジタル信号処理
開催日時 : 12月17日(水)~19日(金)
開催場所 : 高度ポリテクセンター(千葉県千葉市)
受講料 : 30,000円
問い合わせ先: 雇用・能力開発機構 高度ポリテクセンター事業課, ☎(043)296-2582
http://www.apc.ehdo.go.jp/

C言語ベースのシステムLSI設計
開催日時 : 12月18日(木)
開催場所 : CQ出版セミナールーム
受講料 : 13,000円
問い合わせ先: エレクトロニクス・セミナー事務局, ☎(03)5395-2125, FAX(03)5395-1255

JPEG2000動画像符号化技術とビデオストリーミング
開催日時 : 12月19日(金)
開催場所 : オームビル(東京都千代田区)
受講料 : 52,500円(1口で1社3名まで受講可)
問い合わせ先:(株)トリケップス, ☎(03)3294-2547, FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/c031219n.htm

開催日, イベント名, 開催地, 問い合わせ先の順

日程はすべて予定です. 問い合わせ先にご確認のうえ, お出かけください.

シニアエンジニアの技術草子 参拾四之段

◆一万年後に何を残すか？


旭 征佑

●「マガジン　ポン！　わたしにも写せます」のCM

耳にタコができるほど聞かされたこのCMは，家庭用8ミリカメラ「フジカシングル・エイト」のものだ．全盛期はコダックの「スーパーエイト」と合わせ，一般家庭の10%にまで普及したらしい．シングルエイトとは，撮影用8ミリフィルムをマガジンと呼ばれるカートリッジに押し込み，誰でも簡単に扱えるようしたものだ．そうはいいながらもフィルムなので，撮影した後は現像に出す．戻ってくるのは，映画館よろしくリール（円形の枠）に連続撮影されたフィルムを巻いたものだ．

そんな8ミリフィルムのライブラリが数十年ぶりに見つかった．少し引っ張り出して明かりにかざしてみると，たしかに写っている．もちろんフィルム全体を見ないと何が写っているかわからない．タイムカプセルを開いているようなわくわくした気分になって，急きょ試写会を開くことにした．奥のほうで埃をかぶっていた映写機をなんとか探し出し，おそるおそる電源を入れた．幸運なことに，何とか動きそうだ．しかし，ことはそんなに単純ではなかった．

まず，リールの取り付け場所が2か所ある．どちらにフィルムの入ったリールをセットするのかがわからない．次にフィルムをセットする場所が何か所もある．フィルムの先端をどの順番で入れていくのか……，説明書など，とうになくなっている．何とかフィルムをセットし終わったのだが，今度は操作方法がわからない．つまみ類に説明書きがいっさいない．つまみには色が塗ってあり，赤丸がついていたり，緑色だったりする．適当に回したり押したりしながら，何とか操作法を理解した．

試写会が始まると，待ちきれなくなってTVを見ていた家族が再び集まってきた．そして，そこで再現された映像は数十年前の懐かしい記録を鮮やかに映し出し，しばし感慨にふけることとなった．見終わっても皆無言で，その場を動こうとしない．

そんな感動もまださめない頃，困ったことに気が付いた．このままでは，機械が壊れれば二度と見られなくなってしまうではないか．ダビングするといってもそう単純ではない．縦横比が違うことはさておいて，秒あたりのコマ数が8ミリの場合，18コマ/秒だが，通常のTV・ビデオ映像は30コマ/秒だ．映写機の画面をビデオで撮影すると，大きなフリッカが発生する．インターネットで調べたら，「テレシネ」というダビングサービスがあることがわかった．8ミリフィルムをVHSに変換してくれる．10分で5000円以上が相場のようだ．古い記録が蘇るわけだから，決して高くはない．さっそく，依頼することにした．

●「パスポートサイズ」……

そういえば，浅野温子のCMで有名だった8ミリビデオカメラももっていた．押入れの中を何とか探し当てると，文字どおり冷暗所に大量の8ミリビデオテープがひっそりと眠っていた．2時間テープにして30本以上，優に60時間を超える分量だ．さっそく見てみようと思ったが，我が家の8ミリビデオデッキは，「おじいさんの時計」よろしく，もう動かない．ソニー製の8ミリビデオカメラにいたっては，電源コードも電池も見つからず動せない（残念なことに，どちらも専用コネクタなので代用もできない）．既存製品で8ミリビデオデッキがないかと，いくつかのメーカーのWebページを探したが，すでに市販されているものはない．ということは，既出のテレシネのようなサービスを利用して，変換を依頼するしか方法はなさそうだ．量が膨大なだけに，依頼するとかなりの費用がかかることが見えている．せめて，中身を確認してから変換を依頼したいものだ．

8ミリビデオがなくなったのは仕方がないのかもしれない．VHSテープが市場を占有し，何とか残っていたビデオカメラ市場もディジタルビデオに取って変わられたからだ．早いうちにVHSにダビングしておけばよかった．などと思ったが，そんなVHSも，消え去る時期が近い．VHSビデオとDVDレコーダのシェアは2005年度には逆転するといわれている．ビデオカメラも，DVDカメラや，SDカードなどを内蔵したMPEGムービーカメラが登場してきている．いろいろ考えると，今なら，DVDに残しておくべきなのだろうか…….

● 新しい媒体が次から次へと登場する

DVDには，相変わらず多くの規格があり，ご存知のとおりユーザーの混乱をまねいている．現在，開発中の次世代DVDは，松下・ソニーなどのBlu-ray Disc規格と東芝・NECのAOD規格が提案されている．次世代DVDの規格統一をうたうDVDフォーラムでは，規格統一が遅れたまま，製品化そのものはすでに始まっている．2003年度内ぐらいが，本当に規格統一できるかどうかの目安になるだろう．ビデオ，LD，初代DVDと過去のあらゆる局面で規格が分裂し，多くのユーザーが苦し



んだ．先人の轍を踏むことなく，事を進められるかどうか見ものだ．

筆者も技術者のはしくれである．自分が独自に研究した成果が製品として世に出るうれしさは十分に知っているし，素晴らしい革新的な製品であればあるほど，消費者の受ける利益も大きいはずだと自負する気持ちもわかる．経営的判断としても，オリジナル製品のほうが企業として利益率が高いことは一般的な常識だ．

しかし，こんなに多くの媒体が登場し，次々と媒体が変わってしまうのはなぜかと考えると，結果としての判断の甘さに疑問と怒りが込みあげてくるのだ．統一されていない規格は媒体だけではない．実際にCDやDVDでビデオを作成すると，ビデオコンテンツのフォーマット，圧縮形式，MPEG-2のパラメータ設定，音声データのフォーマットとパラメータがやたらと多く，きわめて使いづらい．また，ちょっとした違いで，他の環境ではまったく再生できないということも，日常茶飯事だ．たとえば大事な記録をDVDに残したとしよう．DVDのエンコード方式がわからなくなると，せっかく残したDVDも再生できないなどということが十分に考えられる．

● 新技術が生み出す新しいディジタル化手法

アナログデータは，より忠実に再現するためにどんなに改良が施されようと，データの形そのままを表現している．だからアナログデータは，「データ」さえ残せば，画質が落ちようと，何だろうと再現は簡単にできる．しかし，ディジタルデータは，形あるものを分解し，ディジタル化して保存する．したがって，ディジタル化の手法が改善/改良されれば，同じものでもデータはまったく違う形になる．そのディジタルデータを再現するためには，「ディジタル化手法」と「データ」の両方を保存せねばならない．

ディジタル化手法は次々と改良され，変化していく．そう考えると，ディジタル技術は，時空間を越えて，限りないデータの非再現性を実現してしまっているのではないだろうかという疑問もわいてくる．もしかしたら，データを長く残すというテーマを忘れた開発がなされているのではないだろうか．たとえば，ディジタルデータの一定区域に，共通フォーマットでデータ形式，圧縮形式などの各種情報を列記していくとか，



データ形式の登録機関を設けて管理させることで後日の再現性を高めるとか，できないのだろうか？　そうすれば，あらゆるディジタルコンテンツは，ヘッダ情報を読んだり管理機関に照会するだけでデータを忠実に再現するための情報を得ることができる．

読者は，フランスのラスコー洞窟，スペインのアルタミラ洞窟を覚えておられるだろうか．紀元前1万年以上前といわれる洞窟の中には，たくさんの動物の壁画が今もなお美しい輝きを放ち続け，多くの人に感動を与えつづけている．

現在，書物や美術品などをハイビジョンで撮影してDVDに残す作業が進められているらしい．これらの画像は，同じように1万年後に解読できるのだろうか．1万年たったとき，もしかしたら，われわれの時代だけ何の資料も残らず，歴史の空白になってしまっている，なんてこともありうる．読者だったら，1万年後に何を，どうやって残すだろうか．

あさひ・しょうすけ　テクニカルライター
イラスト　森　裕子

Engineering Life in

エンジニア達の健康管理・なぜエンジニア達は太る？（第一部）

最近の当コラムで行った対談は，シリコンバレーのジムで出会った方々にゲストとして登場していただいた．健康に気を使っている方々ばかりで，もちろん運動も大好きという感じだった．しかし，その一方でアメリカ人の半分ぐらいが太りすぎで，さらにその40%が肥満というニュースもある．アメリカ人はとにかく食事の量が多く，びっくりする．

では，一体エンジニア達はどうやって健康管理をしているのか？　または，していないのか？　今回はこの点について考えてみたい．

☆ 太る原因は大量・大盛り・ビッグサイズにあり

シリコンバレーのエンジニアは人種もそれぞれでさまざまなタイプがいるが，一般的にいえば，やはり日本と比べると巨漢もしくは太った人が多いようだ．太る理由が食事にあるのは大方理解いただけると思う．旅行や出張でアメリカに来られた読者の方々は頷(うなず)くと思うが，アメリカでは食事の量が日本と比べてはるかに多い．食べ物や食事については，このコラムの1998年10月号でも紹介したので，そちらもぜひ参照してほしい．

ごく普通のアメリカ人の食事生活は，どう見ても量が多いのと，太りそうな食べ物……たとえば甘い物や揚げ物が多い．朝からちゃんと食べるのが良いとは日本でもいわれるが，アメリカの「ちゃんと食べる」は半端ではない．卵3個，ハッシュドポテト，ソーセージかベーコン数本，それにトースト2枚，オレンジジュース……これが典型的なアメリカンブレックファーストである．トーストの代わりにホットケーキ3枚またはベーグル，ワッフルを添えるのも立派なアメリカンブレックファーストだ．もちろんホットケーキやワッフルには，大量のバターとシロップが必須となる．量的にいうと日本の2人前は軽くあり，もしかしたら3人前ぐらいはあるかもしれない．

なぜここまで糖分や炭水化物が多く，過激なまでにカロリーを摂取する食事なのか？　筆者なりに考えてみたが，その昔，アメリカ人のほとんどは地方に住み，農業・酪農で生計を立てていたわけである．この名残りでカロリーが非常に高い食事を取るのだと思う．アメリカでいう「軽い朝食」といえば，コンチネンタルブレックファーストと呼ばれる物だ．パン類，オレンジジュース，コーヒーの3点セットが典型的である．しかし，これにもアメリカなりの特徴がある．パン類といってもほとんどがドーナツやケーキに近いものが多いので，とても甘く，もちろんカロリーも高い．それにサイズも大きいから，毎日食べていると太るのは間違いないだろう．

甘い物を食べるのも大きな特徴で，朝昼晩とどこかに必ず甘い物を食べるし，夜寝る前にアイスクリームを食べる習慣があるアメリカ人家庭も多い．また，アイスクリーム屋に行って普通のアイスクリームを頼む際，日本の3倍ぐらいの量がこちらの1人前だと理解しておいたほうがよい．しかも普通のアイスでは寂しいのか，チョコレートのソースやホイップクリームを沢山かけたアイスクリームサンデーを大人が平気で食べることも多い．家庭で自家製のアップルパイを作っても，一緒にアイスクリームと出すパイアラモードにするのが粋なおもてなしなわけだ．日米で存在するドーナツ店などを比較しても，アメリカのドーナツのほうがはるかに甘く感じるというし，実際，日本のものはそれほど甘くないようだ．

☆ 食べ放題が人気？

一方，ランチはというと，たとえばファストフードの場合，日本でも馴染(なじ)みのあるセットメニューがあるが，飲み物一つ取ってみてもアメリカのMサイズが日本のLサイズぐらいだろうか？　セットメニューでよくあるのが「スーパーサイズ」というもので，セットメニューを注文すると，店員から「お客様，1.50ドルプラスすればスーパーサイズにできますが，どうしますか？」などと聞かれる．スーパーサイズにするとドリンクがLよりも一回り大きいサイズになり，フレンチポテトもLよりも一回り大きいサイズになる．ちなみにポテトのスーパーサイズは日本のLの2倍ぐらいの大きさになる．

そのほかに人気のある手軽なランチというと，ピザやメキシカンのブリトーなどがある．いずれにしろ脂っこいし，カロリー抜群だ．中華や和食も人気なのだが，日本では想像できないようなメニューが人気だ．たとえば，和食だと最近では回転寿司やバイキング方式の寿司屋がシリコンバレーでも増えた．人気ネタは海老天やうなぎを太巻きのようにして，それをさらにテンプラの衣をつけて揚げた巻き物だ．すし飯を小さめのおにぎりサイズにしてその中に具を入れて，テンプラにして揚げたようなものだ．1オーダで4個ぐらいになる．試しに食べてみたが，たしかに非常に腹持ちが良く，一気にお腹が一杯になりそうなメニューだった．

食べ放題のお店に人気があるが，和食以外だとインド料理や韓国焼肉の食べ放題が存在する．中華でも安いファストフード系がシリコンバレーでは多くあるので，エンジニア達もお昼や夜に利用することが多い．一度揚げた豚肉や鶏肉を辛いソースに絡めたものが非常に人気で，それを山盛りのチャーハン，もしくは五目焼きそばと一緒に食べるわけだ．もちろん飲み物はコーラや清涼飲料などが多い．日本のファミレスであるようなドリンクバー，炭酸飲料水の飲み放題がやはりよろしいとされる．

「健康的」と思われがちのサラダもお昼の人気メニューだ．ここでもやはり食べ放題のレストランが流行る．サラダだけでなく，数種類のスープやパスタ，そして店内で焼かれている数種

H. Tony Chin

類のパンがすべて食べ放題だ．しかし，いかにも太りそうなこってりとしたドレッシングを沢山かけたり，パンにたっぷりとバターをつけて食べているエンジニア達が多く，少し矛盾しているような気もする．普通のちゃんとしたレストランでも量が多いのには変わりはない．全般的にいえるのが「量が多いのは良いこと」として捉えられていることだろうか？

☆ ジャンクな食べ物は立派な食事である（?!）

清涼飲料といえば，アメリカでの消費量が世界一らしい．とにかくアメリカでは何かにつけて「飲み物はどうしますか？」と聞かれる．もてなすほうとしては，アイスティでも何でも飲んでもらわないと申し訳ないようだ．

それ以外でも，炭酸飲料があらゆる所で飲まれているし，子供もかなり小さいころから飲み慣れている．これは，炭酸飲料も立派な食事の一部と考えられているからだと思う．普通のコーラでも大体ご飯一杯分ぐらいのカロリーに相当するので，かなりのカロリー摂取量となる．これを朝から飲むエンジニアは多いし，お昼でまたもう1本……そして午後の会議中にまたもう1本……となると太るに違いない．

最近，中学や高校で炭酸飲料の自動販売機を置くのは子供達の健康に良くないという理由でジュースとボトルウォーターに代えられているが，このジュースも意外とカロリーが高いので，結果的には太る原因になり，矛盾が指摘されている．

現在では炭酸飲料などは，ジャンクフードということで学校から撤去されたわけであるが，ポテトチップスなどの日本でいうスナック菓子やチョコレートなどは立派な食べ物の一部として日常で捉えられているのがアメリカの食生活だ．たとえば子供のお弁当にハムサンドに小さい袋に入ったポテトチップスを付けることが多い．これにデザートのチョコレートとフルーツを付ければごく一般的なアメリカの子供のランチになる．そのようなことから，街中でサンドイッチ屋に行ってもサンドイッチを食べながらポテトチップスを食べる習慣は大人になっても続くことになる．単なるオヤツではないわけだ．会社に行くとチョコレートやポテトチップスなどが並ぶ自動販売機が設置されているか，スタートアップだとまとめ買いされていたりする．忙しくなるとスナック菓子やチョコレートでしのぐエンジニアが多いので，かなり重宝されている．この数年では日本のカップ麺なども非常に一般的になってきた．もちろんアメリカナイズされた味付けのタイプであるが，会社で数種類，常備していることが多い．

共働きや忙しいエンジニアが多いので，夜も家でじっくり夕食を作る人達は少ない．ここでまたファストフードやレストランのテイクアウトのお世話になる．ちゃんとしたレストランでもテイクアウトに対応しているところがかなりあるので，グルメな食事も事前に電話予約をすれば帰宅時にテイクアウトできる．きちんとバランスの取れた食事をしている人は別として，お昼とあまり変わらない高カロリーな食事をしている人達も多い．おまけにデザートを食べる習慣があるので，ついつい甘い物に手が出てドンドン太る原因を作っていく．

ちなみに夜の付き合いで飲んだりすることは日本ほどない．飲むのが好きなエンジニア達は，シリコンバレーに数か所ある地ビール屋兼レストランに同僚や仲間達と行くことが多い．ここでも美味しいビールとビールに合いそうな食事があるが，やはり揚げ物などが中心になるだろうか？　ビール屋さんではおつまみが半額か無料になる「ハッピーアワー」がコストパフォーマンスが良いことから人気である．

☆ 矛盾している健康食品？

心臓病や循環器系の病気などが注目されていることから，アメリカでも食べ物に気をつけるよう盛んにいわれている．そこで健康に良い食べ物が注目される．日本でもあるような「低脂肪」や「低カロリー」食品だ．カロリーオフの炭酸飲料もたくさん飲まれている．びっくりするのが，豆乳や豆腐でできているアイスクリームだが，なんでも低脂肪なので身体に良い……として売られている．このほかにもどこかピントが外れているものも多い．たとえば，果物や野菜を多く摂りましょう……ということで果物をミキサーにかけたフレッシュジュース屋がシリコンバレーでは流行っている．しかし，冷たくドロッとしたフラッペ状にするため，なんと，かなりの量のシャーベットや甘いヨーグルトが入っている．また，できたジュースもファストフードのLサイズでしか売られていない．せっかくの果物もシャーベットにすることで健康的な意味があまりないように感じるのだが，「ダイエットのためにお昼はこれだけ」というエンジニアも多い．まあ，ファストフードの揚げ物や脂っこい物を食べていないから気休めになるかもしれないが…….

次回の予告

大量に食べたり，甘い物を好むシリコンバレーエンジニア達がどうやって健康を維持するかの努力，そして考え方などについて話を続けたい．

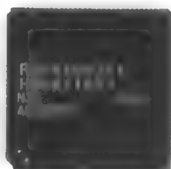
トニー・チン　htchin@attglobal.net　WinHawk Consulting

●32ビットCISCマイコン

H8SX/1657F

- 大容量768Kバイトのフラッシュメモリを搭載し，1サイクルアクセスが可能．
- 複雑な制御を行う大容量システムプログラムを格納でき，プログラムの高速処理を実現．
- 1チップでシステムを構築でき，オンチップでの書き換えが可能．
- H8SX/1650とピン配置が互換．
- 周辺機能として内蔵しているDMACは，「拡張リピートエリア機能」，「オフセット加算機能」などをもつ．
- 演算器および内部バス幅が32ビットのCPUコア「H8SX」を搭載し，35MHzの最大動作周波数で，35MIPS（Dhrystone 1.1）の性能を実現．

●サンプル価格：¥1,700



■（株）ルネサス テクノロジ
TEL：03-5201-5276

●車載用1チップマイコン

M32C/84グループ M32C/85グループ M32C/86グループ

- 最大動作周波数が32MHzで，最小命令実行時間が31.2nsを実現する「M32C/80」CPUコアを搭載．
- 32ビットの演算回路や転送命令，バレルシフタによるシフト命令など，処理性能の向上が図られている．
- 車内LAN規格であるCANコントローラを，最大2チャネル内蔵．
- ISDNなどの通信機器や産業機器で使用されている，HDLC通信対応の回路を2回路内蔵．
- 8チャネルのインプットキャプチャと8チャネルのアウトプットコンペア，UART，クロック同期シリアルI/Oで構成してタイマ機能と通信機能をあわせ持ったインテリジェントI/Oなどの周辺機能を内蔵．

●サンプル価格：¥3,000（84/85グループ）
¥3,500（86グループ）



■（株）ルネサス テクノロジ
TEL：03-5201-5219

●32ビットEthernet CPU

NS9750

- 統合ソリューションの中核となるハードウェア部分「NET+ARM」ファミリの新製品であり，ARM926EJ-Sコアを採用したフラッグシップモデル．
- DSPコード，Javaバイトコードアクセラレータ，MMUの活用が可能な最大200MHz動作可能な32ビットプロセッサ．
- PCI/CardBus，USB2.0（ホスト/デバイス），I²C，IEEE1284（パラレル），シリアル，GPIOなどの幅広い標準の周辺機能をワンチップ化．
- 全二重の10/100Base-T Ethernetは，ワイヤスピードをターゲットとし，またシステムのアプリケーション処理においても高性能を発揮．
- 100MHzのメモリコントローラを搭載し，LCDコントローラを内蔵．
- UART，HDLC，SPIマスタ/SPIスレーブの四つのマルチファンクションシリアルポートをサポート．
- 27チャネルの高速DMAエンジンを搭載．

●価格：下記へ問い合わせ

■ネットシリコン ジャパン（株）
TEL：03-5428-0261　FAX：03-5428-0262

●次世代携帯電話向けアプリケーションプロセッサ

SH-Mobile V2

- DSP機能を持つ高性能CPUコア「SH3-DSP」を搭載し，最大動作周波数は133MHz．
- 200万画素に相当するUXGA対応のカメラインターフェース，高速化および低消費電力化を図ることができるMPEG-4のフルハードウェアアクセラレータおよび2次元/3次元描画の処理を行う2D/3Dグラフィクスエンジンなど，画像処理を強化した機能を搭載．
- 携帯電話システムで必要と見込まれる新規機能の周辺モジュールを内蔵．
- 各種液晶モジュールの特性に合わせて色変換が可能なカラーマネージメントユニットやTFTカラー液晶に対応したLCDコントローラのほか，ビデオ出力ユニットやIrDAインターフェース，最大64和音の音源などの周辺モジュールを内蔵．
- マルチメディアアプリケーション用に，MPEG-4やJPEG，MP3などのほか，豊富なミドルウェアを用意．

●価格：¥4,000（10,000個時）

■（株）ルネサス テクノロジ
TEL：03-5201-5234

●ギガビットEthernetソリューション

DP83864

- 第5世代Ethernetトランシーバを動作させるGigPHYTER Vをコアに採用．
- ギガビットEthernet用トランシーバとしてはユニークな，オンチップマイクロコントローラICを採用．
- システム設計の際に，迅速なカスタム化や高度な設計が可能．
- ソフトウェアの修正だけで，LEDマルチ制御，ケーブル診断や，IP電話用ディテクトなどの新しいアプリケーションの追加が可能．
- フル機能をもつギガビットポートを4ポート装備しており，各ポートは10Base-T，100Base-TXおよび1000Base-TのEthernetプロトコルをサポート．
- IEEE802.3に準拠しており，MII，GMII，RGMIIまたはSGMIIのいずれかのインターフェースを介して，直接MACとインターフェースでき，L2からL4のスイッチ設計に適する．
- IEEE802.3uオートネゴシエーションおよびパラレルディテクション機能を装備．

●価格：下記へ問い合わせ

■ナショナル セミコンダクター ジャパン（株）
TEL：0120-666-116

●LVDS18ビットSerDes

DS92LV18

- LVDS 18ビット・シリアライザ/デシリアライザ（SerDes）．
- パケットを利用したり，パラレルバス周波数を上げたり，追加部品を使用しなくても「非データ」情報を効率的にリンクを通して伝送．
- レシーバのランダムロックにより，トレーニングパターンや特殊なキャラクタが不要．
- ホットインサーション時のシステム干渉がない．
- 15～66MHzの広い動作周波数範囲で，0.270～1.188Gbpsの負荷をサポート．
- 他のSerDesデバイスと比較し，クロッキングへの要求を軽減．
- 送信チャネルと受信チャネルが完全に独立しているため，上りと下りのデータレートを別々に設定できる．
- アイドル動作時や不使用時に，送信部と受信部の消費電力低減が可能．
- ラインおよびローカルループバックテストモードをサポート．

●価格：下記へ問い合わせ

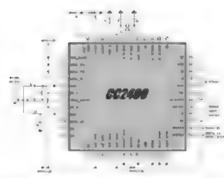
■ナショナル セミコンダクター ジャパン（株）
TEL：0120-666-116

●RFトランシーバ

CC2400

- ノルウェーのChipconAS社が開発した，2.4GHz帯1チップRFトランシーバIC.
- 独自のSmartRF03テクノロジ技術をベースにCMOS 0.18μmプロセスで開発.
- EN300440およびFCC CFR47 Part15，ARIB-STD-T66規格に準拠.
- 受信動作時標準23mA，送信動作時標準19mAの低消費電流.
- ディジタルRSSI出力機能を搭載.
- FSK，GMSKデータ変調機能.
- プログラマブルデータ転送速度は，10K/250K/1Mbpsの3種類を用意.

●価格：¥420（5,000個時）
¥67,900（開発ツール）



■ テクセル（株）
TEL ： 03-5467-9273
E-mail ： chipcon@teksel.co.jp
URL ： http://www.teksel.com/

●ハンドヘルドメディアプロセッサ

GoForce 2150

- 低消費電力なメディアプロセッサで，1.3Mピクセルのカメラエンジンをサポート.
- 64ビットの2Dグラフィックスアクセラレータ，LCDフレームバッファ用内蔵メモリ，CPUインターフェースなどを装備.
- アクティブマトリックスカラースクリーンと，背面に搭載された小型LCDの高速スイッチングが可能.
- 70種類以上のディスプレイインターフェース（CSTN，TFT，OLED，LTPSテクノロジなど）を最大解像度HVGA（320×480）でサポート.
- カメラ付き携帯電話など，ディジタルカメラ内蔵のハンドヘルド端末に対して，高解像度の写真と動画（モーションJPEG）やカメラ制御を組み込むことが可能.

●価格：下記へ問い合わせ

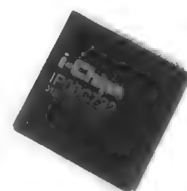
■ エヌビディアコーポレーション
TEL ： 045-640-6850

●カラー画像用解像度変換LSI

IP00C722 (SCREEN-S3)

- ドットマトリクス型の表示デバイスに必要な，カラーディジタル画像の拡大/縮小を1チップで実行.
- NTSCサイズ～SXGAまで幅広い画像入力に対応でき，1個の外付けフレームメモリで，拡大/縮小やフレームレート変換などの機能を実現.
- YUV4:2:2入力に対しては，輝度の10ビット処理に対応し，高画質に画像を表示することが可能.
- RGB24ビットノンインタリーブ，108MHzの画像入力ポート.
- RGB30（24）ビットノンインタリーブ，80MHzの画像出力ポート.

●価格：下記へ問い合わせ



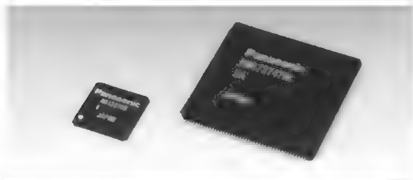
■ アイチップス・テクノロジー（株）
TEL ： 06-6492-7277　FAX ： 06-6492-7388

●映像ディジタル変換チップセット

MN673747 MN673747HL

- 全世界の放送方式（NTSC/PAL/SECAM）に対応したビデオ信号のデコード，3次元Y/C分離，TBC，フレームシンクロナイザを集積し，共通基板でグローバル展開を実現.
- コントラスト/ブライトネス調整回路，色相/色飽和度調整回路，輪郭補正回路，ガンマ補正回路などの画質補正機能を搭載.
- 機能制御用マイコンインターフェースとして，16ビットパラレルインターフェースに加え，I²Cバス制御回路を搭載.
- 10ビット，27MHz動作のA-Dコンバータを3個内蔵し，R/G/B信号からY/Cb/Cr信号への変換回路を搭載することで，3チャネル入力に対応.

●サンプル価格：¥3,000



■ 松下電器産業（株）
TEL ： 075-951-8151
E-mail ： semiconpress@scd.mei.co.jp

●MPEGエンコードチップ

CS92688 CS92689

- 「CS92688」は，これまで別基板が必要だったディジタルビデオ（DV）/IEEE1394メディアアクセスコントローラを内蔵しており，低コストなカムコーダ用入力をサポート.
- クローズドキャプションとEPGをサポートするVBIプロセッシングを集積化.
- コンテンツ保有者の著作権を保護するCPRMをサポートするハードウェアを搭載.
- 「CS92689」は，ディジタルビデオやオーディオレコーディングを可能にするだけではなく，ハードディスクやオプティカルドライブへの記録用にIDEインターフェースを追加.
- 低ビットの圧縮レートでも高品質なビデオレコーディングを可能にし，標準規格のディスクメディアに最大8時間までのコンテンツ録画が可能.

●価格：下記へ問い合わせ

■ シーラス・ロジック（株）
TEL ： 03-5226-7378　FAX ： 03-5226-7677

●フラッシュメモリコントローラ

GBDriver RA3 シリーズ

- 8Gビットまでの NAND型フラッシュメモリを，最大8個まで制御可能.
- ATA規格標準，CompactFlash規格標準のそれぞれに設定可能.
- 独自のフラッシュメモリ制御技術により，バースト書き込みで6.0Mバイト/s，ホストへの読み出しで7.0Mバイト/sを実現.
- 後天的な原因により発生する不良セルへの書き込みを抑止するメモリ制御方式により，フラッシュメモリへのデータ書き込みに対する信頼性を向上.
- 不安定な物理ロジックを検出した場合の動的な判定処理が組み込まれている.
- ビット誤り現象に対して，読み出し時にコントローラ内部で自動的にエラーを修復し，ホスト側のコマンドプロトコルに影響を与えない機能などを装備.
- 小型ロープロファイルタイプの8mm角VFBGAパッケージを採用.

●サンプル価格：¥1,000

■ TDK（株）
TEL ： 03-5201-7102

■弊誌では新製品に関するニュースリリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：（03）5395-2127，E-mail： mngnews@cqpub.co.jp　　（編集部）

●マルチプロトコルシリアルインターフェースポートー

LTC2847/LTC2845

- 「LTC2847」はデータおよびクロック信号用に構成可能な三つのドライバおよびレシーバ，ケーブル終端，チャージポンプを内蔵しているため，チップセット全体が単一電源で動作可能．「LTC2845」はオプションのローカルループバック，リモートループバック，テストモード信号などの制御信号用に構成可能な五つのドライバおよびレシーバを内蔵．
- RS-232-C，RS449，EIA530，EIA530-A，V.35，V.36，X.21をサポートする，ソフトウェアで選択可能なトランシーバ．
- ソフトウェアで選択可能なケーブル終端をサポート．
- 単一5Vの電源動作．

●サンプル価格：¥2,150～（1,000個以上）



■ リニアテクノロジー（株）
TEL：03-5226-7291　FAX：03-5226-0268

●LEDコントローラ

PCA953x/PCA955x

- 携帯電話からコンピュータや通信，ネットワークのサーバまで，幅広い製品でLEDを調光できるように最適化されている．
- ホストインターフェースはI^2Cで，基本的な汎用I/OまたはCPUを使用するだけで，複雑なLED減光や点滅を行うシステムを構築できる．
- 四つの8ビット（256値）内部レジスタ搭載のオシレータを内蔵しており，2種類のプログラム可能な点滅速度の設定が可能．
- CPUが内蔵タイマのいずれかを使用し，反復コマンドを送信して各LEDのオン/オフ切り替えを繰り返す必要がない．
- 標準のGPIOを使用してLEDを調光する場合と比較して，I^2Cバスのデータ量を大幅に削減．
- プログラムを行うことにより，I^2Cバスを切断しても，内蔵オシレータによりLEDはそのまま減光や点滅を継続する．

●価格：下記へ問い合わせ

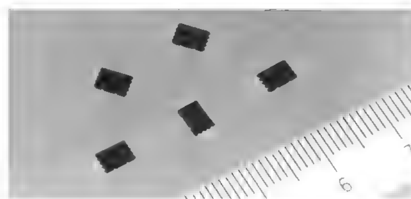
■ ロイヤル フィリップス エレクトロニクス
E-mail：semiconductors.japan@philips.com

●リアルタイムクロック

S-35180A/35380A

- 0.25μAの超低消費電流で携帯機器の長時間動作に適するリアルタイムクロック．
- 動作電圧が1.3V～5.5Vで，メイン電源電圧からバックアップ電源電圧駆動まで幅広く対応可能．
- 1.1Vの最低計時動作電圧によって，バッテリの持続時間を大幅に改善．
- クロック調整機能を内蔵しているため，水晶の周波数偏差を広範囲に補正し，最小分解能1ppmで補正可能．
- クロック調整機能と外部の温度センサを組み合わせ，温度変化に対応したクロック調整値を設定することで温度偏差に対して精度の高い時計機能を実現．

●サンプル価格：¥1,500



■ セイコーインスツルメンツ（株）
TEL：043-211-1193

●A-Dコンバータ

AD9229/AD9289

- シリアルLVDSデータ出力を用いて，4個のA-Dコンバータを1チップ上に集積．
- 「AD9229」は12ビット50/65Msps，「AD9289」は8ビット65Mspsで，高チャネル密度のアプリケーション向け汎用A-Dコンバータ．
- 「AD9229」は，70dBのSNR，85dBcのSFDR，±0.3LSBの微分非直線性（DNL）および±0.6dBの積分非直線性（INL）が特長．3.0V単一電源で動作し，A-Dコンバータコア1個あたりの消費電力は220mW未満．
- 「AD9289」は，47dBの*S/N*比，60dBcのSFDR，±0.5LSBのDNL，および±0.5LSBのINLが特徴．3.0V単一電源で動作し，A-Dコンバータコア1個あたりの消費電力は68mW未満．

●サンプル価格：AD9289 $2.63（1,000個時）
AD9229 $8.50（1,000個時）

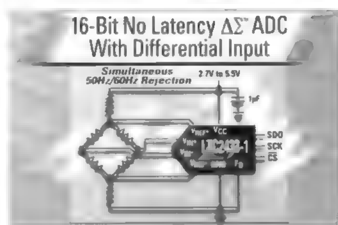
■ アナログ・デバイセズ（株）
TEL：03-3571-5171

●A-Dコンバータ

LTC2433-1

- 1.45μVの低ノイズ性能により，100mV～5Vの広いリファレンス範囲で16ビット性能を提供．
- 100mVのリファレンスによって，±50mVの差動入力信号がPGAなしで16ビットに分解されるため，多くのセンサを直接ディジタル化可能．
- 変換サイクルごとにオフセットおよびフルスケール較正を自動的に行うため，20μVのオフセット誤差，1.25LSBのフルスケール誤差，1.25LSBのINL誤差，0.02LSB以下の遷移ノイズを実現．
- 遅延のない独自アーキテクチャにより，単一サイクルセトリングが可能．

●サンプル価格：¥235（1,000個時）



■ リニアテクノロジー（株）
TEL：03-5226-7291　FAX：03-5226-0268

●D-Aコンバータ

LTC1588/LTC1589

- 「LTC1588」は単調12ビット，「LTC1589」は単調14ビットの電流出力D-Aコンバータ．
- 出力範囲を3線シリアルインターフェースによるプログラムが可能で，二つのユニポーラ範囲（0V～5V，0V～10V）と四つのバイポーラ範囲（±2.5V，±5V，±10V，－2.5V～7.5V）の合計六つの出力範囲のいずれでも動作可能．
- いずれの出力範囲に対しても，±1LSBのINL，±1LSBのDNLが全インダストリアル温度範囲で規定．
- 出力範囲を切り替えるための高精度抵抗，スイッチ，外付けアンプは不要．
- 一つのボード設計で，計測，データ収集など複数のアプリケーションに対応可能．
- 起動および動作に必要な外付け部品は，5Vリファレンス，デュアルOPアンプ，フィードバックコンデンサのみ．

●サンプル価格：
LTC1588 ¥715～（1,000個時）
LTC1589 ¥1,195～（1,000個時）

■ リニアテクノロジー（株）
TEL：03-5226-7291　FAX：03-5226-0268

●ディジタルアンプIC

YDA137

• 最大5W×2チャネルのディジタルアンプIC.
• アナログ信号入力回路，パルス幅変調回路，BTLディジタル出力回路，自励発振回路，過電流保護回路，ポップノイズ低減回路，ヘッドフォンアンプなどディジタルアンプに必要な機能を小型パッケージ28ピンTSSOPに集積.
• 独自の低損失CMOS設計手法により，2Ωインピーダンスで最大出力5Wを実現.
• 独自の回路技術により，同クラスのディジタルアンプと比較して，歪率は約1/2以下の0.02%(1.5W出力時)，*S/N*比は100dB以上の性能を達成.
• 現行のアナログアンプICと比較して，3Ωインピーダンスでの電力効率は最大3倍の約82%，動作時消費電力は最大で1/4，アンプの発熱に相当する電力損失は最大1/7など，高効率，省電力，低発熱性を実現.

●サンプル価格：¥500

■ **ヤマハ(株)**
TEL ： 0539-62-5444

●水晶発振器

TG-5000LA/TG-5001LAシリーズ SG-8002LB/SG-8002LAシリーズ SG-550/SG-350シリーズ

• 独自のプラスチックモールディング技術と小型セラミックパッケージ振動子技術を組み合わせた製品.
• リードフレームを介した一般プラスチック半導体ICと同様の構造となっており，セラミックパッケージ品と比較して，実装基板からのストレス吸収が可能.
• ストレスから周波数変化を生じる特性をもつ水晶発振器で，耐環境特性で高い信頼性を確保.
• ワイヤボンディング内部実装を採用.

●サンプル価格：¥1500
(TG-5000LA/TG-5001LAシリーズ)
¥600(SG-8002LB/SG-8002LAシリーズ
SG-550/SG-350シリーズ)

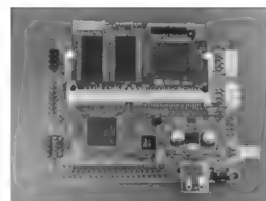
■ **セイコーエプソン(株)**
TEL ： 042-587-5878
URL ： http://www.epsondevice.com/

●1ボードコンピュータ

N-Card

• タンバックのV_R4131DIMMモジュール(TB0229)を用いた，Linuxが動作する1ボードコンピュータ.
• 基板サイズ95×72mmの小型ボードサイズ.
• 取り付けコネクタ位置は，市販のユニバーサル基板にあわせてある.
• USB経由でEthernet，無線LAN，USBメモリなどを使用可能.
• MMCでWindowsとのデータのやり取りが可能.
• 4Mバイトのフラッシュメモリだけで，telnetd，httpdなどが利用可能.
• MMC/USBメモリを利用すれば，さらに多くのファイルを使用可能.

●価格：¥42,000



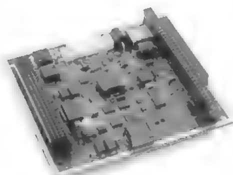
■ **(有)ハンブルソフト**
TEL ： 096-312-3208　FAX ： 096-312-3218

●非絶縁型電源ボード

MAPC-104

• シスコ システムズ社3200シリーズ モバイル アクセス ルータに完全対応.
• 車載用に適する9～32Vdcの広い入力電圧範囲をもつ.
• シスコ システムズ社 3200シリーズカード駆動に必要な，3.3V@6A，5.1V@6.3A，12V@100mAを完全保護付きで供給.
• 3出力DC-DCコンバータボードで，負荷の変動やバッテリの充電程度による電圧変化に対応.
• 90%以上の高い変換効率は，低消費電力，信頼性向上，良好な温度特性を保証.
• IEC/EN/UL6095認定.
• PC/104-Plus規格に対応.

●価格：¥30,400(1～9個時)



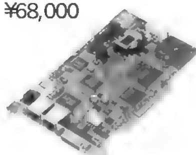
■ **デイテル(株)**
TEL ： 03-3779-1031　FAX ： 03-3779-1030

●RASボード

PC-RAS (PCI)

• Webサーバを搭載しているため，パソコンのスタンバイ電源からの電源供給でパソコンとは完全に独立して動作し，ネットワーク上の独立したノードとして認識される.
• パソコンの動作が不安定な状況下でも，Webサーバや RAS機能は停止することなく稼動を継続.
• ネットワーク上のパソコンから，Webブラウザを使ってアクセスして，ネットワークやRAS機能の設定，稼動状態の監視，本体パソコンの起動/シャットダウンが行える.
• 筐体内温度，ファン回転数，電源電圧をリアルタイムに収集し，ユーザープログラムでの参照やWebブラウザによる遠隔監視が行える.

●価格：¥68,000



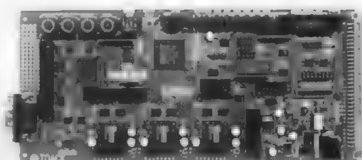
■ **(株)コンテック**
TEL ： 03-5628-9286　FAX ： 03-5628-9344
E-mail ： tsc@contec.co.jp

●ディジタル制御ボード

Rune Blade DSP-002-3U

• アナログデバイセズ社製の160MHz動作「ADSP-2191M」を搭載した，DSPディジタル制御ボード.
• ICE接続用JTAGインターフェースを備えているため，ブート用フラッシュメモリを実装することで，スタンドアロンシステムとして動作させることが可能.
• 標準でディジタルI/O，汎用スイッチ，RS-232-Cを備え，オプションで最大3チャネルの高速A-Dコンバータを搭載可能.
• DSP拡張メモリインターフェースにより，ユーザー回路の追加が可能.
• 5V単一電源で動作.
• 基板形状はユーロボード3Uに準拠し，19インチラックに実装可能.

●価格：¥189,000



■ **(株)トアック**
TEL ： 03-3526-9345　FAX ： 03-5298-6284

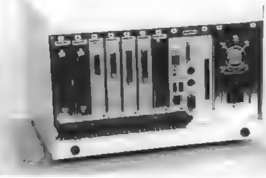
■弊誌では新製品に関するニュースリリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX :(03)5395-2127，E-mail： mngnews@cqpub.co.jp　(編集部)

●リアルタイムシミュレータ

RTSim-C/LT-RTSim-c

- MATLAB/Simulinkで設計したシミュレーションモデルに入出力ポートを追加し，Realtime WorkshopによりCソースコードを自動生成した後に，シミュレーションモデルを実時間で動作させることが可能.
- ハードウェアにはPC/AT互換機を採用.
- CompactPCIに対応し，CPUボード（Pentium Ⅱ：850MHz）および周辺ボードで構成.
- RTSim-CはRT-Linuxを，LT-RTSim-cはPC-DOSを，それぞれOSに採用.

●価格：¥1,600,000（RTSim-C）
¥2,000,000（LT-RTSim-c）



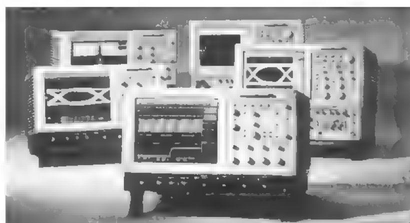
■ ディエスピーテクノロジ（株）
TEL：0533-73-1388 FAX：0533-73-1389
E-mail：rtsim@dsptec.co.jp
URL：http://www.dsptec.co.jp/

●ディジタルフォスファオシロスコープ

TDS7000Bシリーズ

- TDS7404B型は周波数帯域が4GHz，TDS7254B型は2.5GHz，TDS7154B型は1.5GHzのディジタルフォスファオシロスコープ.
- SiGe技術やDPX技術などにより，400,000回/s以上の波形取り込みレートを実現.
- 1台の測定器で回路の特性検証を行い，コンプライアンステストの実行が可能.
- データレートではGbpsオーダ，立ち上がり時間では100psオーダの測定が可能.
- 専用に開発されたICにより，機能を拡張したトリガ機能やクロックリカバリ機能を搭載.

●価格：¥5,840,000（TDS7404B型）
¥4,470,000（TDS7254B型）
¥3,420,000（TDS7154B型）



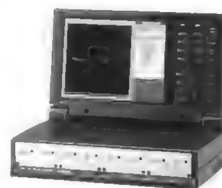
■ 日本テクトロニクス（株）
TEL：03-3448-3010 FAX：0120-046-011

●多チャネルディジタルオシロスコープ

DM3300

- 絶縁で40Mspsの高速電圧アンプを用意し，広帯域計測を実現.
- 電圧/温度アンプ，ひずみアンプ，ロジックアンプを用意し，測定目的に合わせてアンプの着脱が可能なプラグイン方式を採用.
- 同時サンプリングと絶縁型のプラグインアンプで，高精度な測定を実現.
- TCP/IPインターフェースとUSBインターフェースを標準装備.
- 2MW/chのメモリとPCMCIAドライブを標準装備.
- オプションで40GバイトのHDDの搭載が可能.

●価格：¥440,000（8チャネル）
¥470,000（16チャネル）



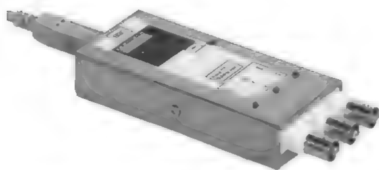
■ グラフテック（株）
TEL：0120-626294
E-mail：graphcs@graphtec.co.jp

●パルスカウンタ

TUSB-S01CN1

- USBインターフェース付きで，データはパソコンの大容量メモリに収納.
- ゲートコントロール，カウンタリセットは，パソコンなど外部からの操作が可能.
- 各入力の正負理論は，パソコンから自由に設定可能.
- 電源は不要で，接続はすべてBNCコネクタで行う.
- 最大カウント周波数は，50MHz.
- 最大カウント数は，4294967296（32ビット）.
- 入力数，ゲートコントロール入力，カウンタリセット入力はそれぞれ1点.
- 最大接続台数は，4台（内部スイッチでID選択，HUBが必要）.

●価格：¥19,800



■（株）タートル工業
TEL：029-843-0045 FAX：029-843-2024

●波形解析ソフトウェア

TDSPCS1

- ディジタルストレージオシロスコープTDS1000/2000シリーズ，ディジタルフォスファオシロスコープTDS3000Bシリーズの波形取り込み，波形解析およびレポート機能を強化するソフトウェア.
- オシロスコープで取り込んだデータをPCで解析し，レポートを作成することが可能.
- ExcelおよびWord用の，TDSオシロスコープツールバーが含まれており，MS-Excel 2000/XP，MS-Word 2000/XPおよびOpenChoiceデスクトップに直接インストールされる.
- ツールバーを使用することで，波形，オシロスコープの設定，測定データなどをワンクリックでインポート可能.
- MATLABやLabVIEWなどの解析アプリケーションなどにデータを送ることができる.

●価格：¥21,800

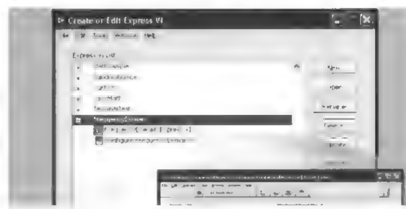
■ 日本テクトロニクス（株）
TEL：03-3448-3010 FAX：0120-046-011

●開発ツールキット

LabVIEW Express VI 開発ツールキット

- 計測，テストアプリケーション開発ソフトウェア「LabVIEW 7 Express」の新機能「Express VI」をユーザーがカスタマイズするためのツールキット.
- アプリケーション作成のための5～15の標準的関数がユニット化されており，マウスによるポイントアンドクリック操作で対話式に機能設定できるため，アプリケーション開発時間の短縮が可能.
- 既存の標準VIやLabVIEW 7 Expressに付属している38種類のExpress VIの修正，Express VIテンプレートをカスタマイズするだけで，Express VIの作成が可能.

●価格：¥150,000



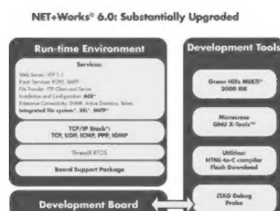
■ 日本ナショナルインスツルメンツ（株）
TTEL：03-5472-2970 FAX：03-5472-2977
E-mail：prjapan@ni.com

●組み込みネットワーク機器向け統合ソリューション

NET+Works 6.0

- Ethernetをベースとしたコネクティビティを実現．
- 組み込みネットワーク機器開発に必要な開発ツール，ボード，デバッガ，ランタイム環境，ネットワークプロトコル，アプリケーションなどをワンパッケージ化．
- SSL，WiFiドライバ，ACE，フラッシュファイルシステム，SNTPなどの機能を追加．
- NET+ARMファミリに最適化されたThreadXリアルタイムOS，フルTCP/IPスタック，組み込みネットワークに必要なプロトコルやユーティリティを幅広くサポート．

●価格：下記へ問い合わせ



■ ネットシリコン ジャパン（株）
TEL：03-5428-0261　FAX：03-5428-0262

●開発言語

REALbasic 5 日本語版 for Windows

- C++やJavaのような最新のオブジェクト指向を取り入れた，BASIC言語によるプログラム作成ツール．
- GUIオブジェクトやデバッガ，クロスプラットホームのコンパイラなどを搭載した開発環境．
- Macintosh同様，Windowsでも容易にプログラムを開発する環境を提供．
- 30種類以上のパーツが用意されており，ドラッグ&ドロップで簡単にウィンドウやダイアログのデザインが作成可能．
- 完成したプロジェクトは，コンパイルすることにより独立したアプリケーションにでき，本体やランタイムモジュールなしで動作可能．
- Windows版で作成したプロジェクトは，コンバートなしにMacintosh版で編集が可能．
- Professional版では，Macintosh上で動作するアプリケーションの開発も可能．

●価格：¥17,000（Standard版）
¥59,000（Professional版）

■（株）アスキーソリューションズ
TEL：03-4524-6001
E-mail：retail@asciisolutions.com
URL：http://www.asciisolutions.com/

●ドキュメント自動作成ツール

A HotDocument for ANSI-C Ver1.2

- ソースファイルより20種類以上のドキュメントを自動生成．
- ソースファイル一覧，関数一覧，関数フロー，クラスメンバ関数定義書など納品物件に必要な情報を出力．
- 社内システムの内部資料としても，ソースファイルを解析するリバースエンジニアリングツールとしても利用可能．
- 出力形式は，Excelファイル，テキストファイルの2形式．
- ファイルの指定は，ファイルおよびフォルダ単位でできるため，より多くのソースファイルを指定可能．

●価格：¥39,800



■（株）ハローシステム
TEL：03-5367-5183　FAX：03-5367-5181
E-mail：info@hellosystem.com

●アプリケーション日本語化ツール

Alchemy CATALYST 5.0

- プロジェクト管理機能を備えた，統合ローカリゼーションツール．
- アプリケーションのEXE/DLL/OCXファイルを直接日本語化できるため，アプリケーションのソースコードは不要．
- AlchemyのezMatch翻訳メモリテクノロジにより，一度翻訳した文書を再度翻訳する必要がない．
- Oracle，IBM DB2，MS SQLなどの多くのデータベースタイプに対応．
- ezParseにより，あらゆるファイルタイプのテキストパーサを短時間で開発可能．
- UTF-8，UTF-16，Shift-JISなどの一般的なエンコードシステムをサポート．
- ビジュアルXLIFF翻訳環境を提供．
- 翻訳者が使用するアクティビティと機能を高機能で使いやすいインターフェースに統合．

●価格：¥90,000～

■ エクセルソフト（株）
TEL：03-5440-7875　FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com
URL：http://www.xlsoft.com/

●ホストアクセスソリューション

Reflectionシリーズ Windows版 v11/Web版 v6

- 米国WRQ社が開発した，ホストアクセスソリューション．
- 高度で安全な暗号化や認証方法を提供し，安全性の高い接続で顧客の機密情報を守る．
- 多彩なインストール方法を提供することにより，配布にかかる手間や時間を軽減．
- 提供されるVBAやAPIを使用することにより，非効率なホストアプリケーション操作やシステム構築を改善することが可能．
- Windowsのグループポリシを利用して，システム管理者が組織やグループごとに設定や動作を統一することが可能．
- セキュリティ機能があらかじめ組み込まれているため，SSH，SSL/TLS，Kerberos，XDM-Authorization-1などのホスト接続を，より簡単な操作で利用可能．
- PC Xサーバでは，OpenGL Ver1.4およびGLX Ver1.3をサポート．

●価格：¥21,000～

■ サイバネットシステム（株）
TEL：03-5978-5453　FAX：03-5978-2201
E-mail：rinfo@cybernet.co.jp
URL：http://www.cybernet.co.jp/reflection/

●組み込みLinux用ボードサポートパッケージ

μLinux Consumer Electronics Edition

- 対応ボードは，DBPXA250（インテル）およびMS7727RP02（日立）で，同社が提供するμLinux ELITEに追加することで利用可能．
- 対応カーネルは，Kernel 2.4.20以上．
- 組み込み用の基本パッケージ群をRPMとSRPM形式で提供．
- ツールチェーンとして，GNU C/C++，GNU ASM（gas），GNUリンカ（ld），バイナリユーティリティ，デバッガ（gdb，kgbd）をサポート．
- ネットワークは，IPv4/IPv6に対応．
- ステレオサウンドの再生，録音，エコーキャンセラ機能を装備．
- ハードリアルタイムの実装を実現．
- 割り込み応答は1.64μs，スレッド起動は4.68μs，周期スレッドは15.0μsを実現．
- 省電力に対応し，起動時間を短縮．
- メモリ管理機能の充実．

●価格：下記へ問い合わせ

■ リネオソリューションズ（株）
TEL：03-5730-0123　FAX：03-5730-0125

■弊誌では新製品に関するニュースリリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュースリリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

読者の広場

Interfaceへの声



2003年11月号特集「マイクロプロセッサ技術の基本」に関して

▷前号に引き続いて，キャッシュメカニズムに関してていねいに解説されていたのがたいへん良かったです．命令キャッシュが大きくなると，デバッガでブレークポイントを設定する際，いろいろ制限が出てくるようです．本号のおかげで"ヘネパタ本"が読みこなせそうです．（白石 隆）

▷予告にあった"コンフィギュラブルプロセッサ"など，今後のトレンドの詳細がなかったのが残念!（倉科 輝樹）

［編］たいへんもうしわけありませんでした．原稿はいただいていたのですが，特集全体がオーバフロー気味となり，やむなく掲載を断念せざるを得なかった原稿もあります．機会をみて掲載したいと考えております．

▷私には内容がかなり高度で勉強になります．いまだに10月号と合わせて読み返しています．永久保存版として読み返していきたいと思います．（USB）

［編］プロセッサに関した解説記事は，今後も定期的に取り上げていくつもりです．解説してほしいプロセッサなどがあれば，読者アンケートはがきでどしどしご意見をお寄せください．

Interface全般に関して

▷C#について取り上げてほしい．BorlandからもC# Builderが登場して，C#もC++やJavaと並んでメジャーな言語になってきた．C#とともに，.NETテクノロジが今後のエンベデッドプログラムにどのような影響を与えるか，そのようなニュースも扱ってください．

（組み込まれ型プログラマ）

▷ハッカーの常識的見聞録はタイムリーな記事でした．今度ぜひ「セキュリティの観点からの組み込みシステム」という記事をお願いします．（JR9JUK）

［編］11月号掲載の「ハッカーの常識的見聞録」での話題は，インターネットやメールのウィルス対策/セキュリティ対策でした．今後はインターネット家電/情報家電がどんどん増えていくことが予想されるので，組み込み機器でのセキュリティは，今後ますます重要な要素になっていくと思われます．

特集担当デスクから

☆特集扉を含めると全90ページという大特集号になりましたが，いかがだったでしょうか．スプリットトランザクションもしない，バースト転送にすら対応しないのでは，PCI-Xの意味がない，そんなデバイスを作るのはPCI-Xのポリシに反するのでは？という声もあるでしょう．たしかにおっしゃるとおりです．今回の設計は，ここで紹介した設計をそのまま実際のシステムに使おうというものではなく，より複雑になったPCIバスシステムを理解するための第一ステップとしてとらえてほしいと考えています．

☆TECH I Vol.3『PCIデバイス設計入門』では，まずはじめにコンフィグレーションレジスタをもたない，しかもアドレス固定の書き込み専用PCIデバイスを設計するところからはじめています．従来からあるISAバスなどと考え方が大きく異なるPCIバスを理解するために，そぎ落とせる仕様は徹底的に落とし，これ以上ないくらい簡単なハードウェアにしたうえで，PCIバスを理解するためです．

☆今回の特集企画でも，当初は基本的にTECH I Vol.3と同様の方針で考えていたのですが，現実にはそれが難しい場合があることがわかりました．というのは，実際にPCI-Xを搭載したマシンでは，PCIバス番号が1本だけということはありえず，内部的に複数のPCIバスが実装されています．第5章で解説したように，このようなシステムでは当然ながらPCI-PCIブリッジも存在することになります．このとき，コンフィグレーションレジスタを実装しないPCI-Xデバイスを用意したとしても，システムからはコンフィグレーションレジスタが読み出せない＝デバイスが存在しないと判断され，結果的にPCI-PCIブリッジのウィンドウがディセーブルに設定されてしまうのです．

☆つまり，CPUから目的のPCI-Xデバイスまでの経路が途中で寸断されてしまうため，CPUのアクセスがそのデバイスまで届かなくなってしまうわけです．

☆以上のような理由から，今回はシステムにPCI-Xデバイスとして認識してもらうための最低限の仕様を，当初から実装せざるを得ませんでした．そのため，コンフィグレーションレジスタを実装し，まずはシングル転送だけに対応したデバイスということで，今回設計したPCI-Xデバイスの仕様が導き出されたわけです．

☆PCI-Xを制するには，まずはFRAME#など信号線の制御ルールと，アトリビュートフェーズを理解し，シングル転送を攻略（?）する必要があるでしょう．その次はバースト転送への対応，そしてスプリットトランザクションへも対応したイニシエータ（リクエスタ）機能を実装できれば，PCI-X完全制覇です!?

☆PCI-Expressよりもまず，目の前にある，今すぐ使える高速バスを活用していこうではありませんか!

読者の広場

アンケートの結果

興味のあった記事
(2003年11月号で実施)

①第1章 キャッシュのメカニズム
②プロローグ プロセッサ興亡史
③第2章 MMUの基礎と実際
④第4章 命令セットアーキテクチャの変遷
⑤第3章 割り込みと例外の概念とその違い
⑥Appendix 1 高速化技術の基礎
⑦フリーソフトウェア徹底活用講座(第12回)
⑧組み込みLinuxをとりまく世界(第3回)
⑨組み込みGUI設計の現状とソリューション(第1回)
⑩初級ドライバ開発者のためのWindowsデバイスドライバ開発テクニック(第2回)
⑪Appendix 2 高信頼性をサポートする機能
⑫回路図形式で演算を行えるツール「Try Signal」の概要
⑬SDIOカード開発入門(第2回)
⑭シニアエンジニアの技術草子(参拾参之段)
⑮移り気な情報工学(第35回)
⑯1線式デバイスによるWebベース多点温度計測
⑰開発環境探訪(第23回)
⑱「VxWORKS」を使ったRTOS技術の基礎と応用(第1回)
⑲プログラミングの要(第7回)
⑳ハッカーの常識的見聞録(第35回)
㉑海外・国内イベント/セミナー情報
㉒Engineering Life in Silicon Valley(対談編)

特集『マイクロプロセッサ技術の基本』についてのアンケートの結果

Q1 先月号(2003年10月号)の特集もご覧いただいていますか?

①読んだ(100%)
②読んでいない(0%)

Q2 先月号と今月号の2号連続のプロセッサ解説特集企画はいかがだったでしょうか?(複数回答可)

①各機能が詳しく書かれていて良かった(22%)
②パイプラインやキャッシュの動作など，各機能の動作が理解できた(19%)
③一般論と実プロセッサの実装事例の比較が良かった(22%)
④各プロセッサ別に同じ処理内容のサンプルアセンブラソースの比較があるとおもしろかったかも(8%)
⑤現在の8/16ビットのCPUについても取り上げてほしい(2.5%)
⑥もっと昔のCPUについても取り上げてほしい(0%)
⑦もっとマイナーな組み込みRISC系CPUについても取り上げてほしい(2.5%)
⑧HDLなどで実際にプロセッサを設計してソースを公開してほしい(19%)
⑨2号連続の特集企画はやめてほしい(2.5%)
⑩その他(2.5%)

Q3 今後，特集で取り上げて欲しい分野があれば，教えてください．

- VLIWやコンフィギュラブルプロセッサ
- IA32からIA64，x86-32からx86-64へ
- HDLによるプロセッサの設計解説特集
- グラフィックスチップ
- RISCプロセッサの組み込みにおける応用事例

など

Interface 年間予約購読のお知らせ

Interfaceを確実にお手元にお届けする年間予約購読をご利用ください．

Interface：毎月25日発売
年間予約購読料金：10,800円
※予約購読料金の中には年間の定価合計金額および送料荷造り費用が含まれます．

● 申し込み方法

お申し込みは，FAXで下記までご通知ください．お申し込みに便利な「年間予約購読申込書」をWeb上でも公開しています(http://www.cqpub.co.jp/hanbai/nenkan/nenkan.htm)．こちらもご利用ください．

お支払い方法は，クレジットカード・現金書留・郵便振替・銀行振込がご利用になれます．

お申し込み受け付け後，請求書を発送いたします．

● 年間予約購読の申し込み先

CQ出版株式会社 販売局 販売部
TEL：03-5395-2141　FAX：03-5395-2106



読者プレゼント



●応募方法：本誌読者アンケートはがきに必要事項を記入のうえ，2003年12月31日(必着)までにご応募ください．なお当選者の発表は発送をもってかえさせていただきます．

(1) アルテラロゴ入りTシャツ
(2名)
アルテラ(株)
(http://www.altera.co.jp/)
サイズ：L



(2) IrFront H85 Trial Kit (1名)
詳細はp.142を参照してください．

編集後記

●IQについての番組がありました．四つの図形を示し，五つ目にくる図形はどれかという問題や，図形を五つ示し，ほかの四つの図形と異なる図形を選ぶといった問題です．某外資系コンピュータ会社の入社試験はこのIQ試験でした．はたしてIQはいくつだったのでしょう．Interfaceを引き継ぐことになりました．よろしく．（檀）

●メカ部の調子が悪かった古いビデオデッキがついにお亡くなり．比較的新し目のSVHSデッキ1台のみの状態なので，懸案だったHDD内蔵ビデオレコーダ購入プロジェクトがついに本格化？ M下にするかT芝にするか，Pにやぁ～かSオニーか（笑）話のネタ（？）にCEATECでも見てきたPSXという選択肢も……で，発売日はいつ？（M）

●冷房なしで真夏を乗り切った自宅マシンのハードディスクがお亡くなりに．そこで追悼記念にDVD-MULTIを買ってきて，DVD-Rにデータを退避しています．本当はネットから落としたデータを個人でバックアップする必要はないのですが，いつまでも落とせるとは限らないのがネットの弱点なので今日もR焼き．（み）

●入社以来，通巻162号から今回の319号まで，計158号の間，本編集部で仕事してきました．13年を超えようとするこの秋，「編集委員」として異動することとなりました（担当分野は，いままで＋αという感じです）．いろいろありがとうございました．後任の編集長は，山形孝雄です．今後とも，どうかよろしく，お願いいたします．（洋）

●気が付けばもう1月号です．でもこれを書いているのは11月の初旬だったりして，日付の感覚が完全に狂ってます．さて，しばらくインターフェースの編集に参加させていただいていたのですが，古巣に戻ることになりました．また2週に1回締め切りがやってきます．さらに日付感覚がおかしくなりそうな予感．（@）

●ドライブを兼ねて箱根の仙石原へすすきを見に行きました．晴れていればさぞやきれいだったのでしょうが生憎の雨．残念ながら思い描いていたような見渡すかぎり一面のすすきが風で揺れる景色は見られませんでしたが，山は紅葉が始まっていて素敵でした．たまには季節を感じに出掛けるのもなんだかいいものですね．（Y2）

●先日，紅葉見物に日光いろは坂をドライブしてきました．東北道宇都宮から日光有料へ入る場所から，渋滞が始まっており目的地に向かうのを諦め，霧降高原に方向転換しました．こちらは，車の数が極端に少なく驚きました．ただし，終点には牧場があり大変な混雑をしており，いったい何時に家を出てくるのか驚きの連続でした．（S）

●10月25日にMac OS X（10.3）が発売された．私は購入していないのだが，良い噂と悪い噂を色々と聞く．特にPC業界では出荷されたばかりの物というのはトラブルがつきものであるが，今は好奇心と損得勘定のはざまで揺れ動いているところである．しかし10.2→10.3の0.1のバージョンアップで14,800円は高い．（ふ）

お知らせ

■読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■コピーサービスのご案内

本誌バックナンバーの掲載記事については，在庫（原則として24か月分）のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金（税込み）
  1ページにつき100円
- 発送手数料（判型に関わらず）
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛先
  〒170-8461　東京都豊島区巣鴨1-14-2
  CQ出版株式会社　コピーサービス係
  （TEL：03-5395-4211，FAX：03-5395-1642）

■お問い合わせ先のご案内

- 在庫，バックナンバー，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株）　2004　振替　00100-7-10665
2004年1月号　第30巻　第1号　通巻第319号）
2004年1月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏　村上真紀　山口光樹　落合幸喜　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／神崎真理子　森　祐子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122　FAX／（03）5395-2127
広告部（03）5395-2133　URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141　E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社



日本ABC協会加盟誌
（新聞雑誌部数公査機構）　ISSN0387-9569